5- Transport Layer (10)

Compiled by-UBM

Transport Layer (10)

- **5.1 The Transport Service**: Transport service primitives, Berkeley Sockets, Connection management (Handshake), UDP, TCP, TCP state transition, TCP timers
- 5.2 TCP Flow control (sliding Window), TCP Congestion Control: Slow Start

Chapter Outline

13.1 Transport-Layer Services

13.2 Transport-Layer Protocols

Topics Discussed in the Section

- ✓ Process-to-Process Communication
- ✓ Addressing: Port Numbers
- ✓ Encapsulation and Decapsulation
- ✓ Multiplexing and Demultiplexing
- ✓ Flow Control
- ✓ Error Control
- ✓ Congestion Control
- ✓ Connectionless and Connection-Oriented Services

Figure 13.1 *Network layer versus transport layer*

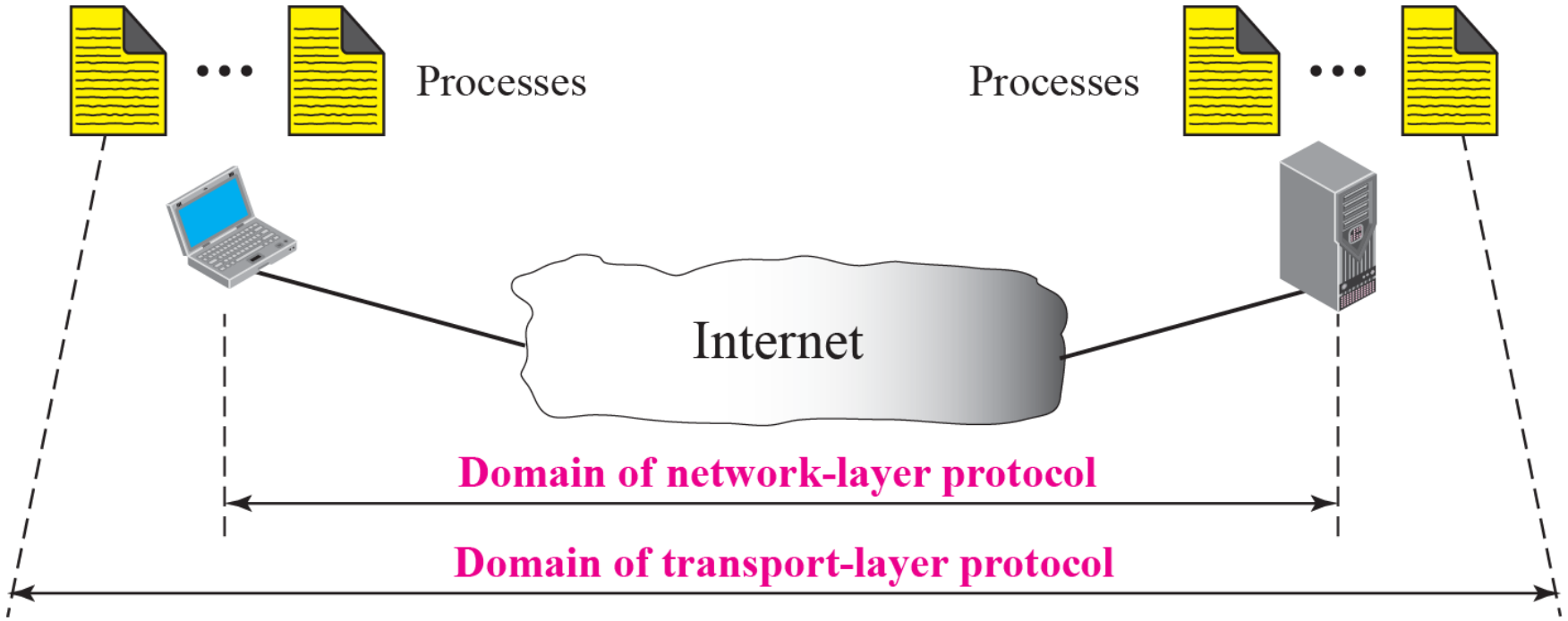


Figure 13.2 *Port numbers*

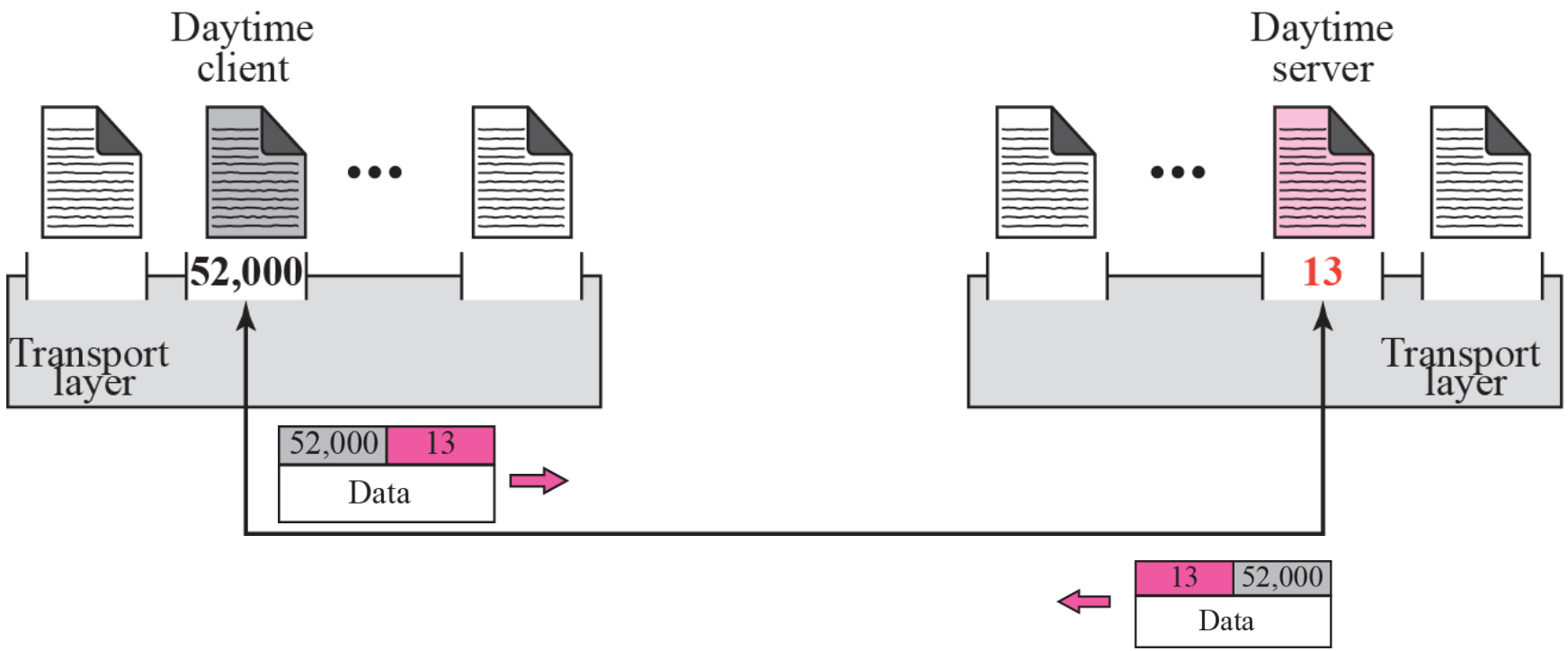


Figure 13.3 *IP addresses versus port numbers*

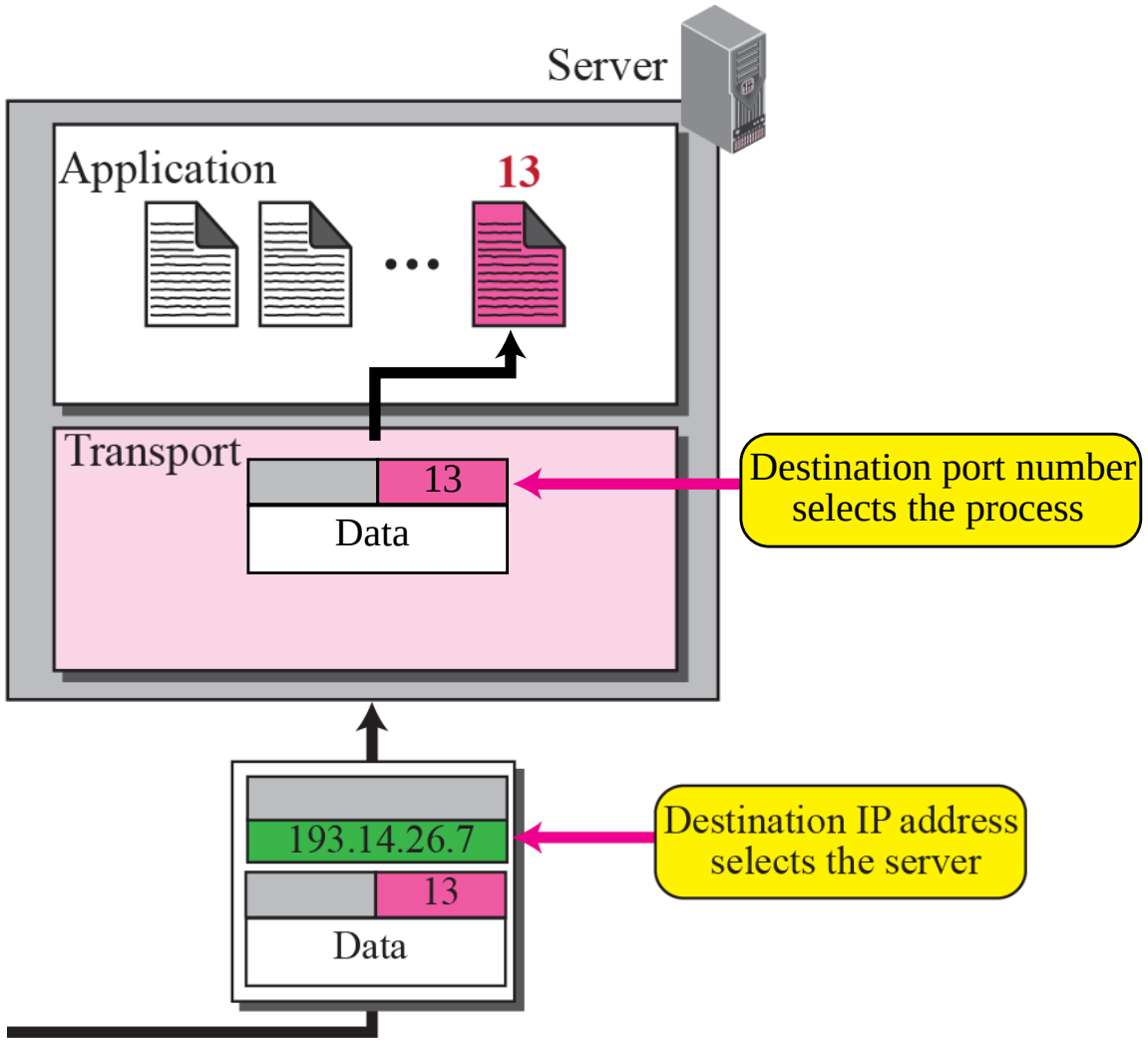
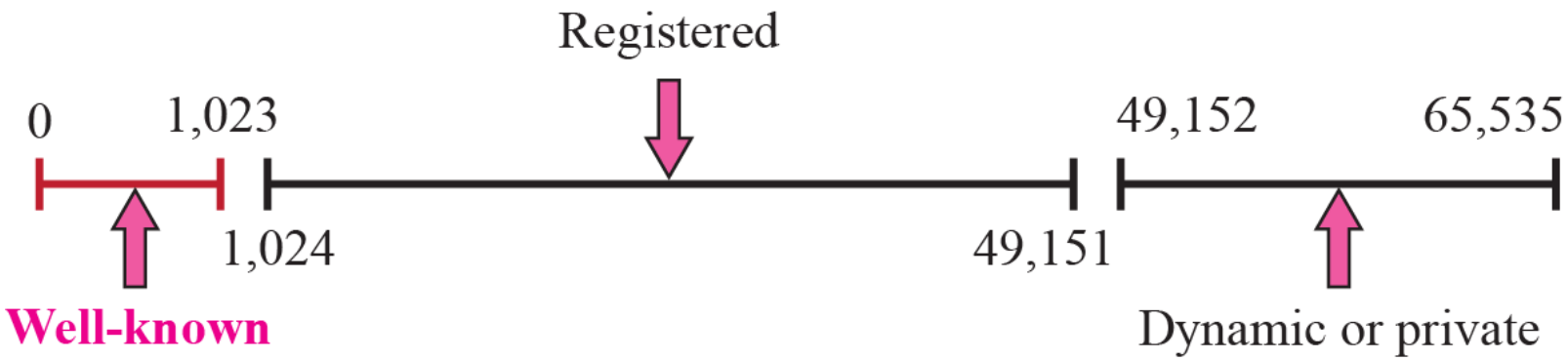


Figure 13.4 *ICANN ranges*



Note

The well-known port numbers are less than 1,024.

Example 13.1

In UNIX, the well-known ports are stored in a file called /etc/services. Each line in this file gives the name of the server and the well-known port number. We can use the grep utility to extract the line corresponding to the desired application. The following shows the port for TFTP. Note that TFTP can use port 69 on either UDP or TCP. SNMP (see Chapter 24) uses two port numbers (161 and 162), each for a different purpose.

```
$grep    tftp   /etc/services
   tftp          69/tcp
   tftp          69/udp
```

```
$grep    snmp /etc/services
   snmp161/tcp#Simple Net Mgmt Proto
   snmp161/udp#Simple Net Mgmt Proto
   snmptrap162/udp#Traps for SNMP
```

Figure 13.5 *Socket address*

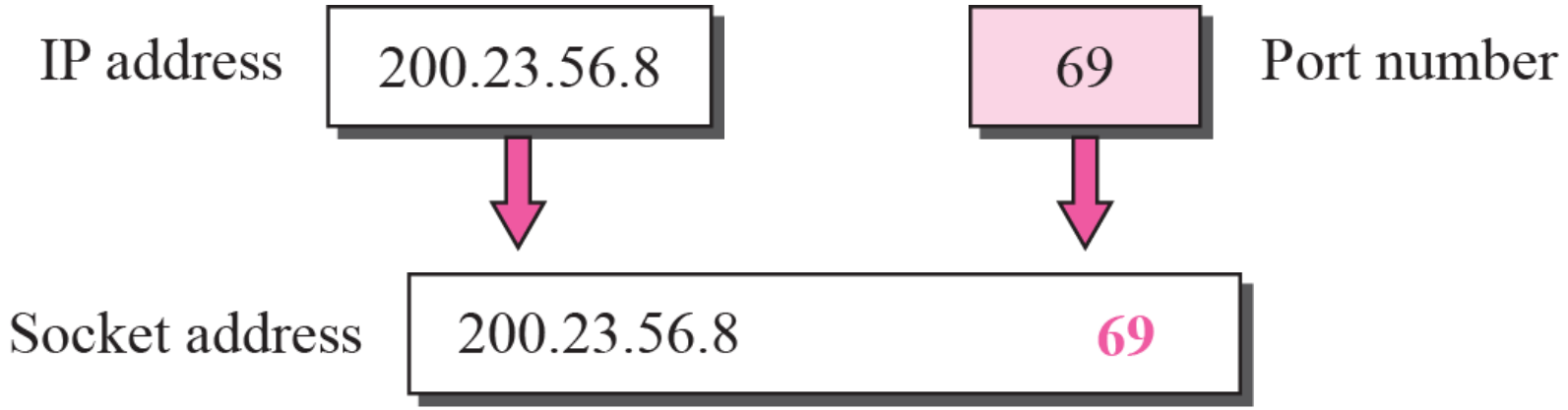


Figure 13.6 *Encapsulation and decapsulation*

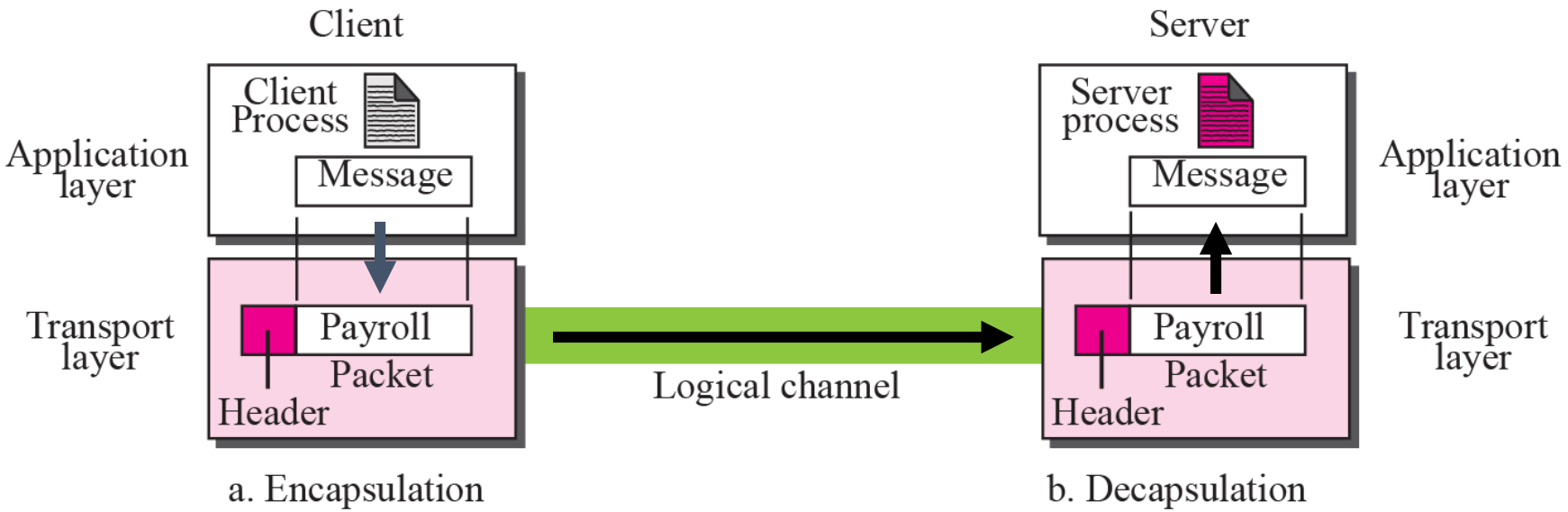


Figure 13.7 *Multiplexing and demultiplexing*

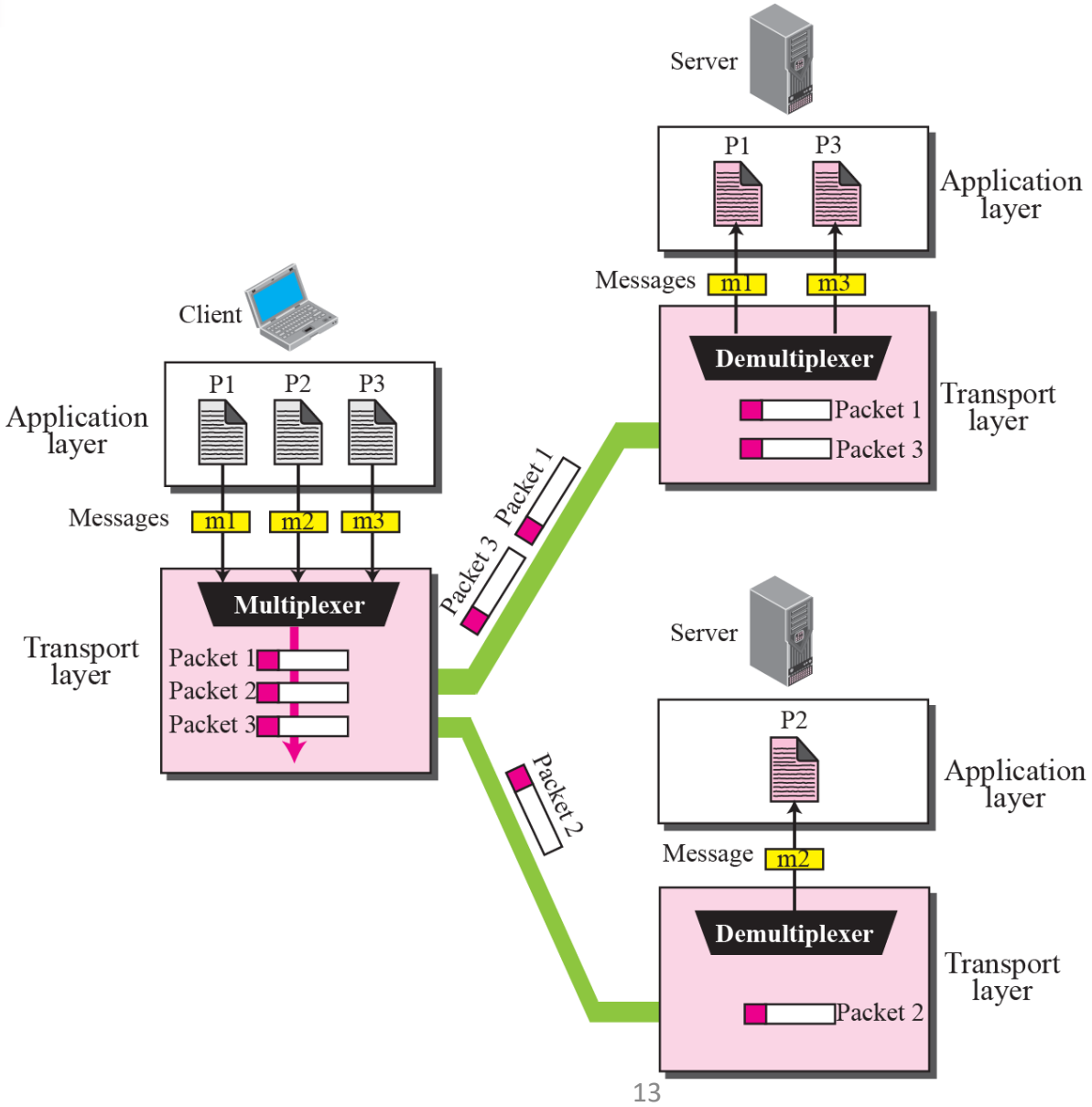
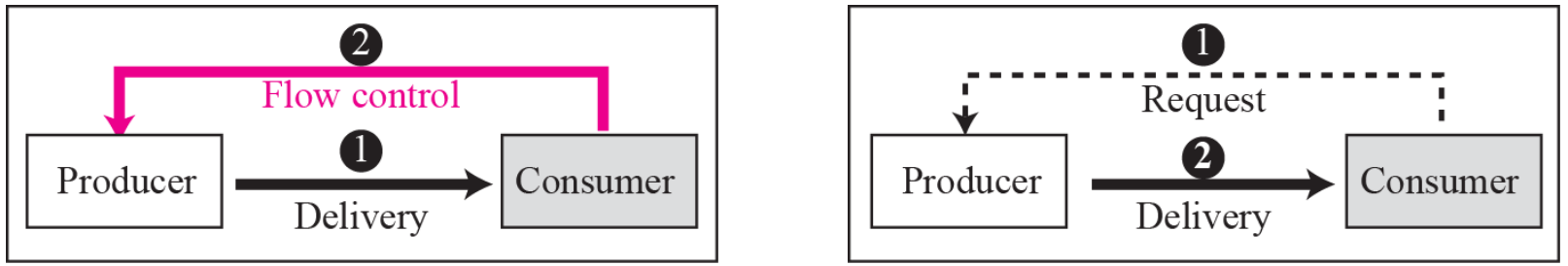


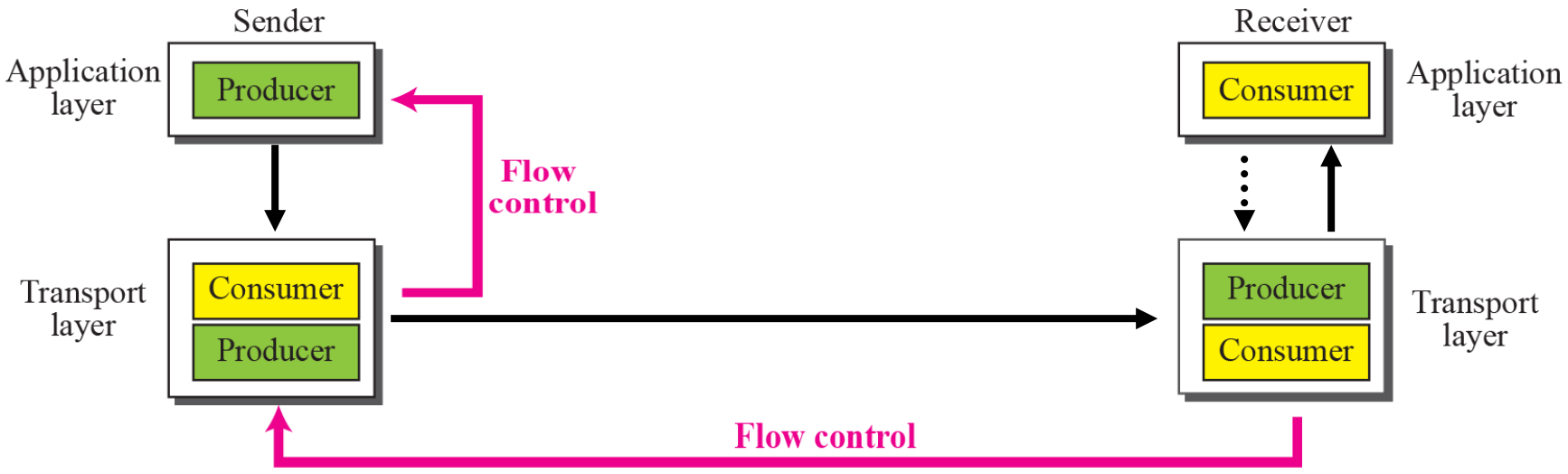
Figure 13.8 *Pushing or pulling*



a. Pushing

b. Pulling

Figure 13.9 *Flow control at the transport layer*



Example 13.2

The above discussion requires that the consumers communicate with the producers in two occasions:

i) when the buffer is full and

ii) ii) when there are vacancies.

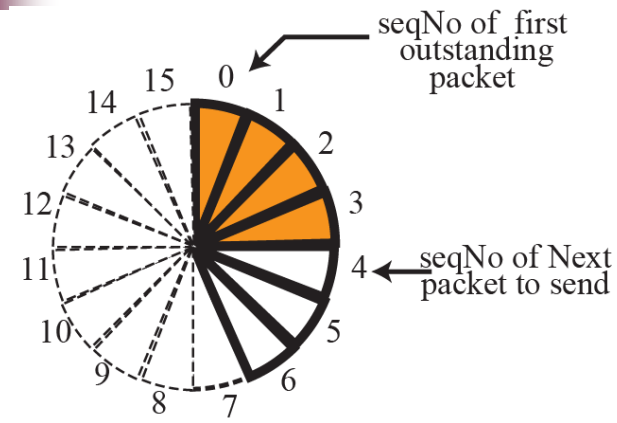
If the two parties use a buffer of only one slot, the communication can be easier. Assume that each transport layer uses one single memory location to hold a packet. When this single slot in the sending transport layer is empty, the sending transport layer sends a note to the application layer to send its next chunk; when this single slot in the receiving transport layer is empty, it sends an acknowledgment to the sending transport layer to send its next packet. As we will see later, this type of flow control, using a single-slot buffer at the sender and the receiver, is inefficient.

Figure 13.10 *Error control at the transport layer*

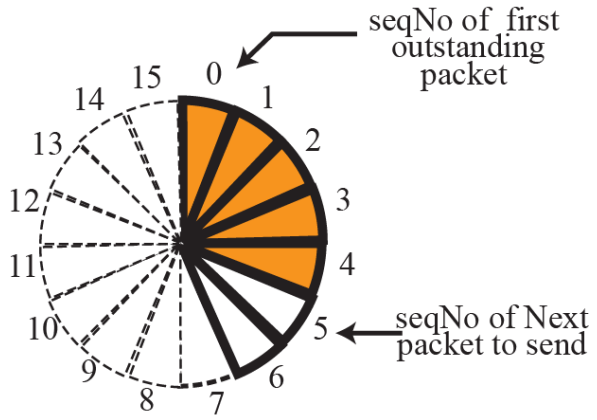
Note

For error control, the sequence numbers are modulo 2^m, where m is the size of the sequence number field in bits.

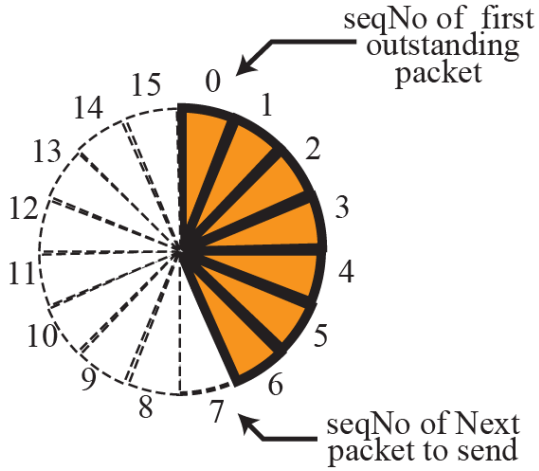
Figure 13.11 *Sliding window in circular format*



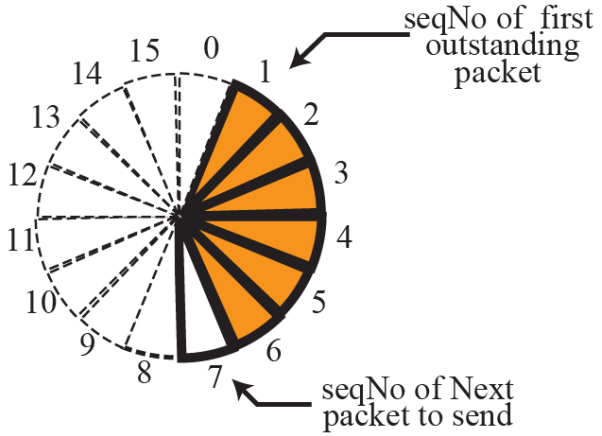
a. Four packets have been sent



b. Five packets have been sent



c. Seven packets have been sent window is full



d. Packet 0 has been acknowledged, window slides

Figure 13.12 *Sliding window in linear format*

a. Four packets have been sent

b. Five packets have been sent

c. Seven packets have been sent
window is full

d. Packet 0 have been acknowledged
and window slid

Figure 13.13 *Connectionless service*

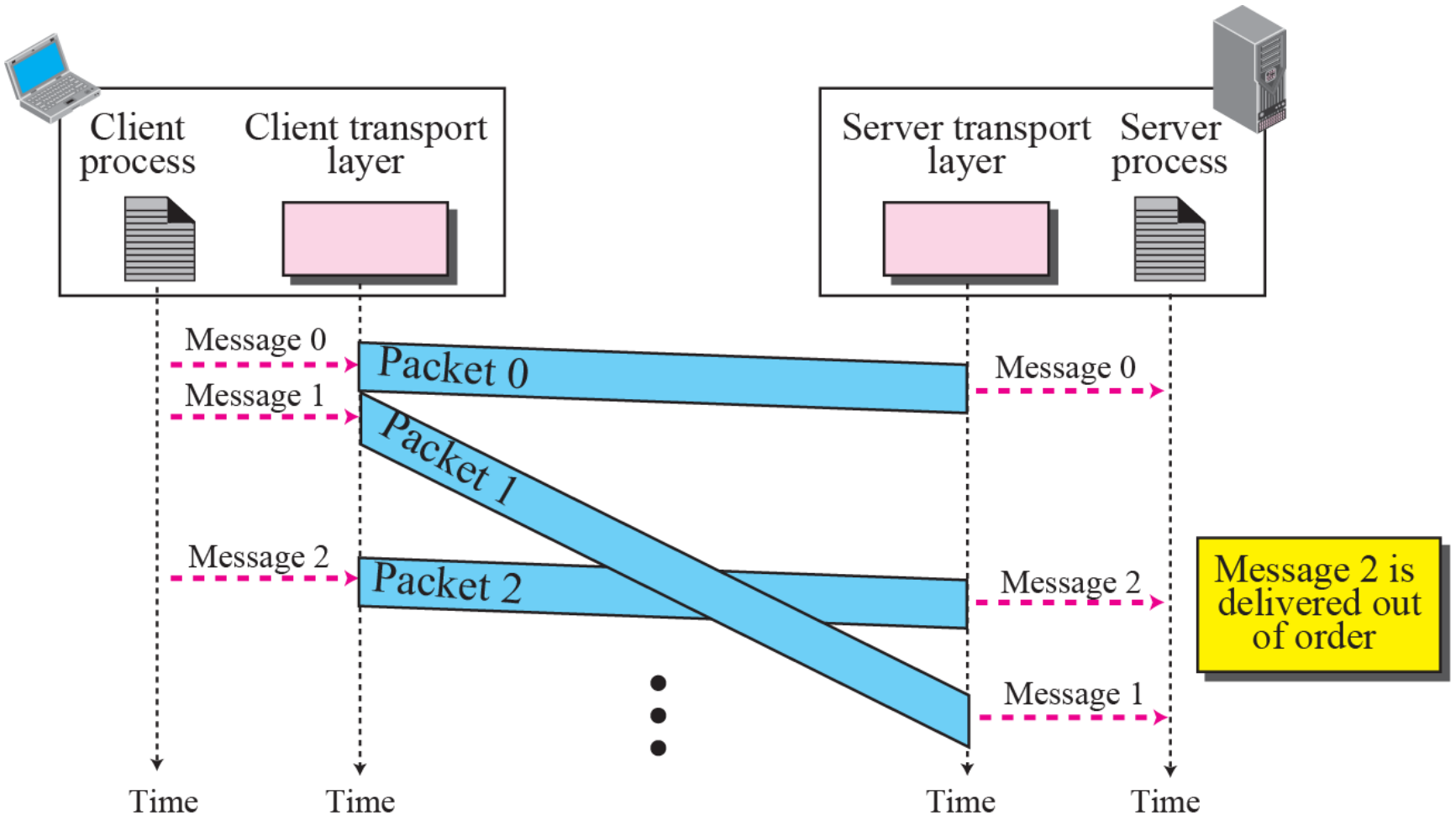


Figure 13.14 *Connection-oriented service*

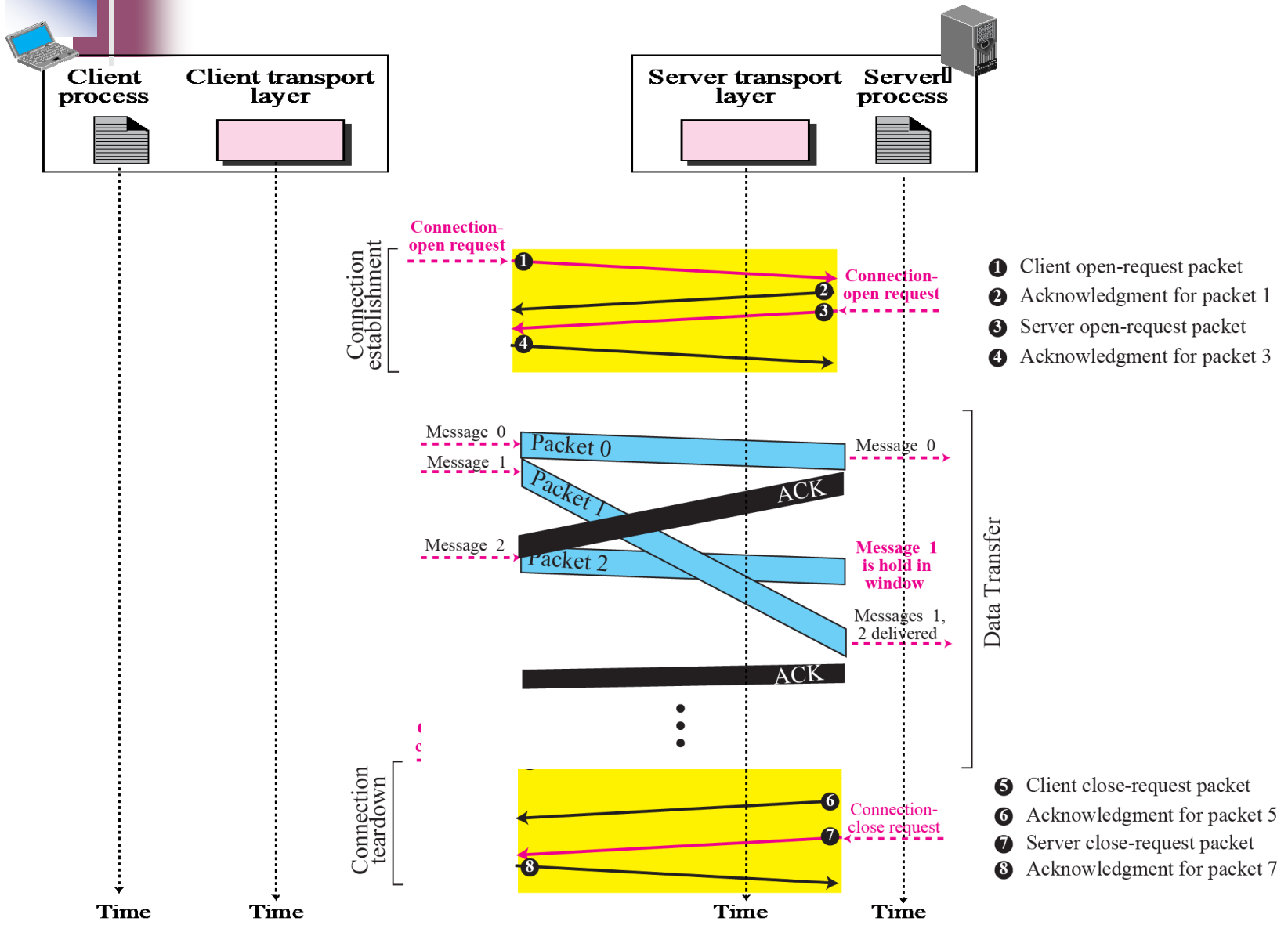


Figure 15.1 *TCP/IP protocol suite*

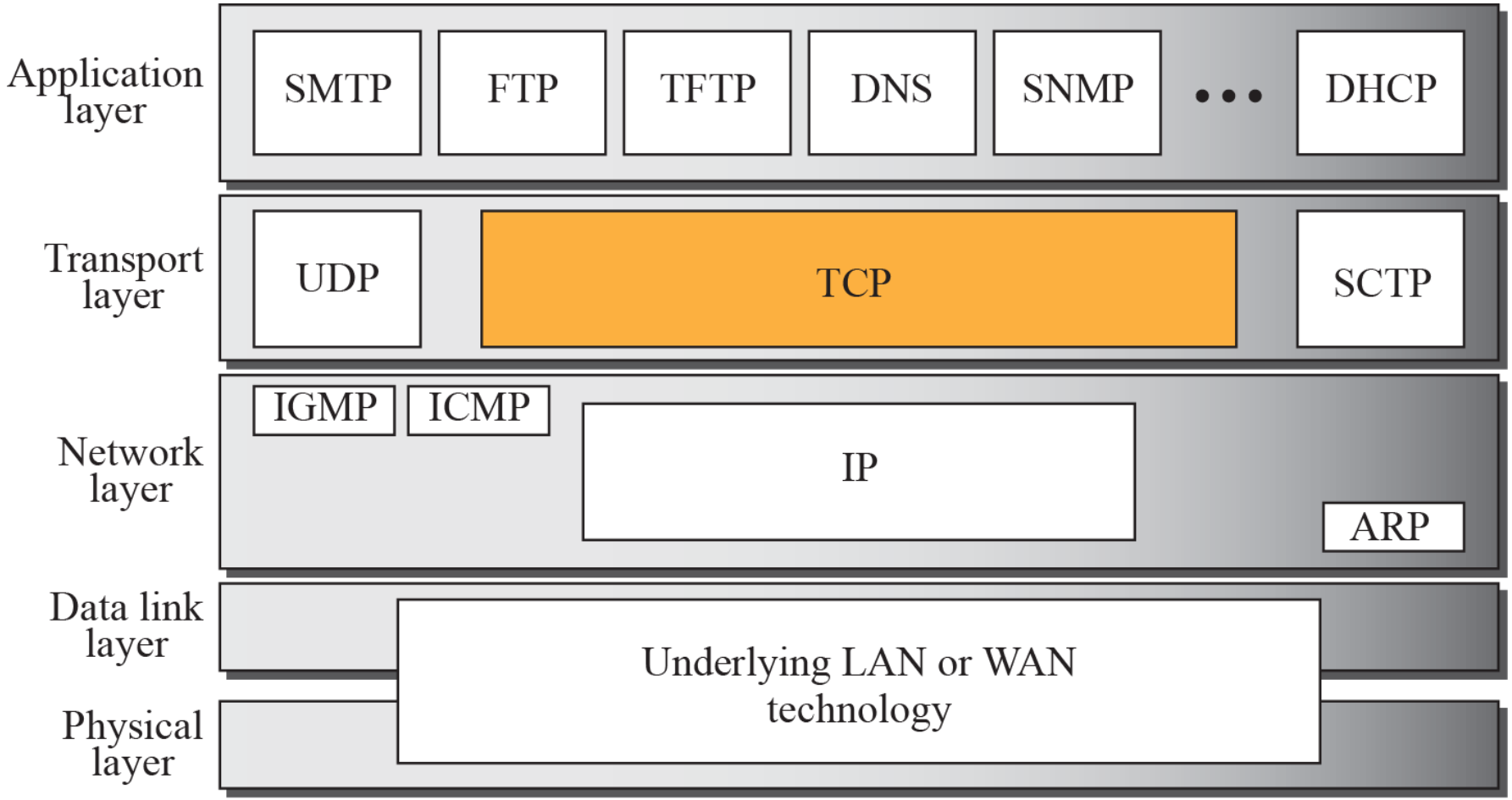


Table 15.1 *Well-known Ports used by TCP*

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
20 and 21	FTP	File Transfer Protocol (Data and Control)
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol

Figure 15.2 *Stream delivery*

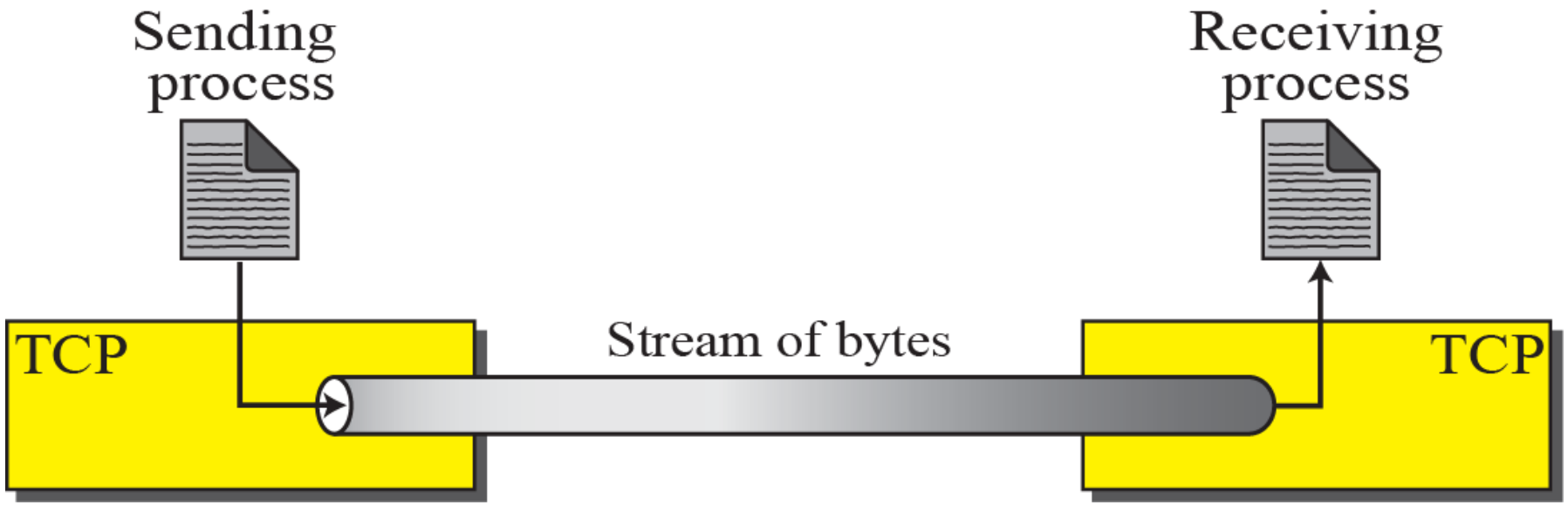


Figure 15.3 *Sending and receiving buffers*

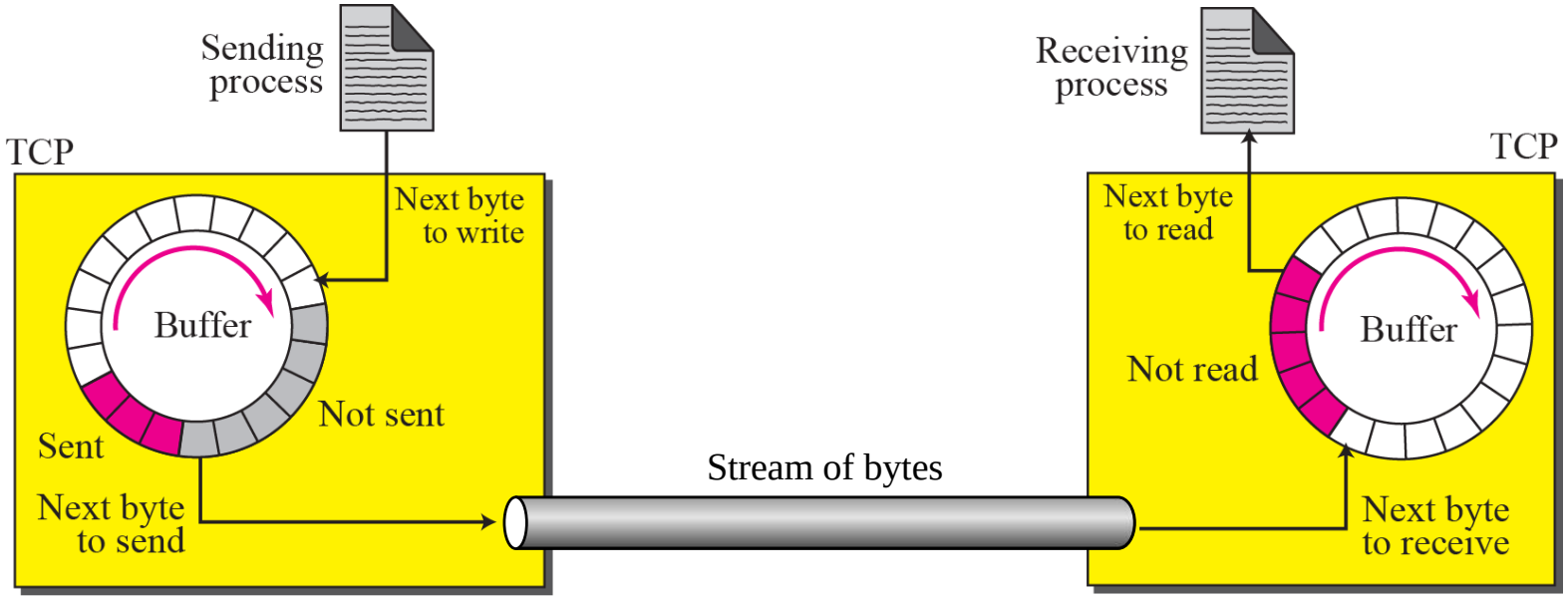
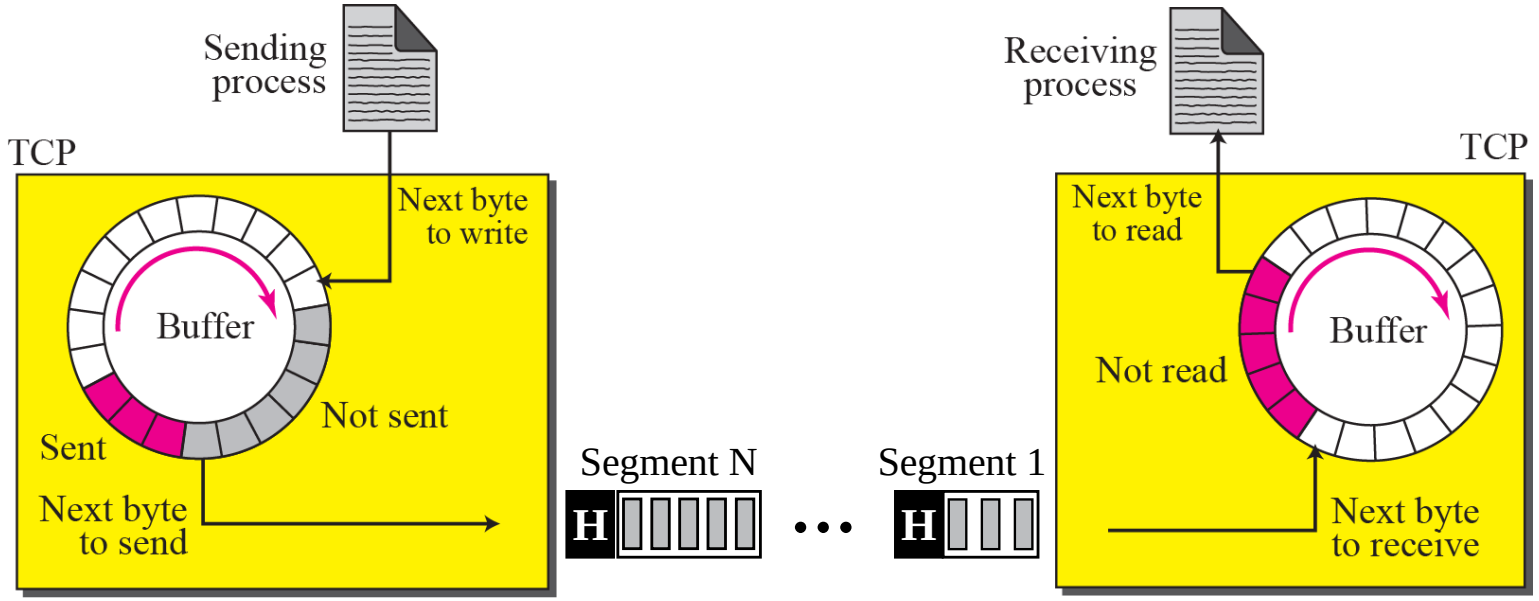


Figure 15.4 *TCP segments*



15-2 TCP FEATURES

To provide the services mentioned in the previous section, TCP has several features that are briefly summarized in this section and discussed later in detail.

Topics Discussed in the Section

- ✓ **Numbering System**
- ✓ **Flow Control**
- ✓ **Error Control**
- ✓ **Congestion Control**

Note

The bytes of data being transferred in each connection are numbered by TCP.

The numbering starts with an arbitrarily generated number.

Example 15.1

Suppose a TCP connection is transferring a file of 5,000 bytes. The first byte is numbered 10,001. What are the sequence numbers for each segment if data are sent in five segments, each carrying 1,000 bytes?

Solution

The following shows the sequence number for each segment:

Segment 1	→	Sequence Number:	10,001	Range:	10,001	to	11,000
Segment 2	→	Sequence Number:	11,001	Range:	11,001	to	12,000
Segment 3	→	Sequence Number:	12,001	Range:	12,001	to	13,000
Segment 4	→	Sequence Number:	13,001	Range:	13,001	to	14,000
Segment 5	→	Sequence Number:	14,001	Range:	14,001	to	15,000

Note

The value in the sequence number field of a segment defines the number assigned to the first data byte contained in that segment.

Note

The value of the acknowledgment field in a segment defines the number of the next byte a party expects to receive.

The acknowledgment number is cumulative.

15-3 SEGMENT

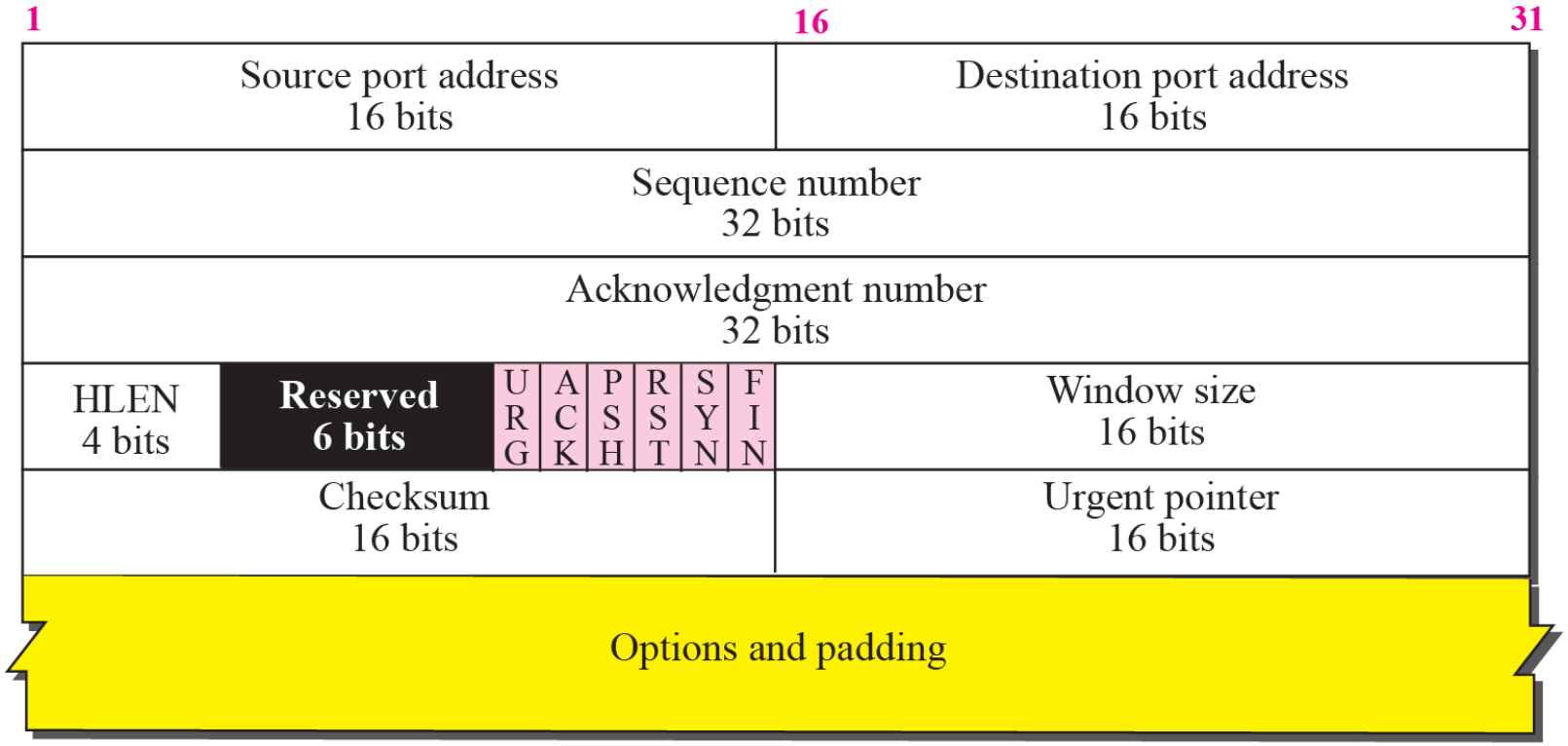
Before discussing TCP in more detail, let us discuss the TCP packets themselves. A packet in TCP is called a segment.

Topics Discussed in the Section

- ✓ **Format**
- ✓ **Encapsulation**

Figure 15.5 *TCP segment format*

a. Segment



b. Header

Figure 15.6 *Control field*

URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH: Request for push
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: Terminate the connection

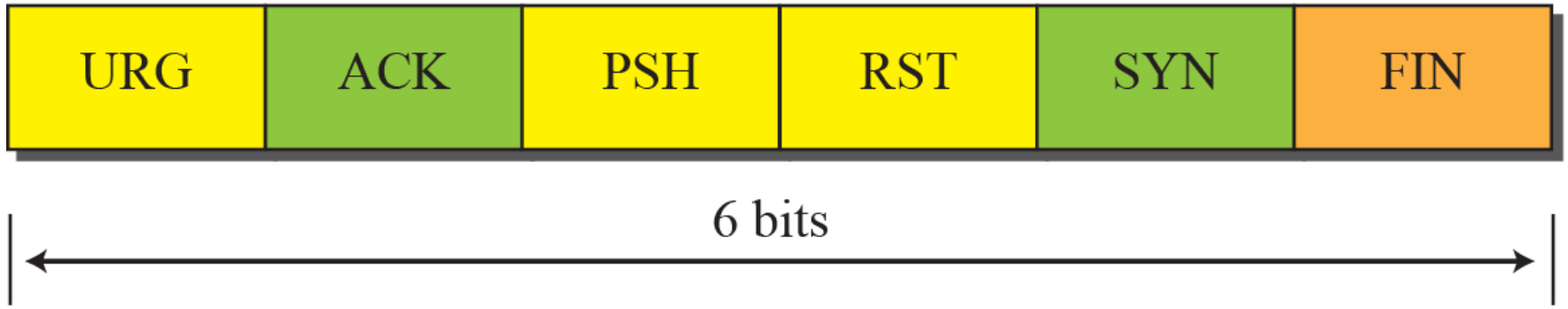
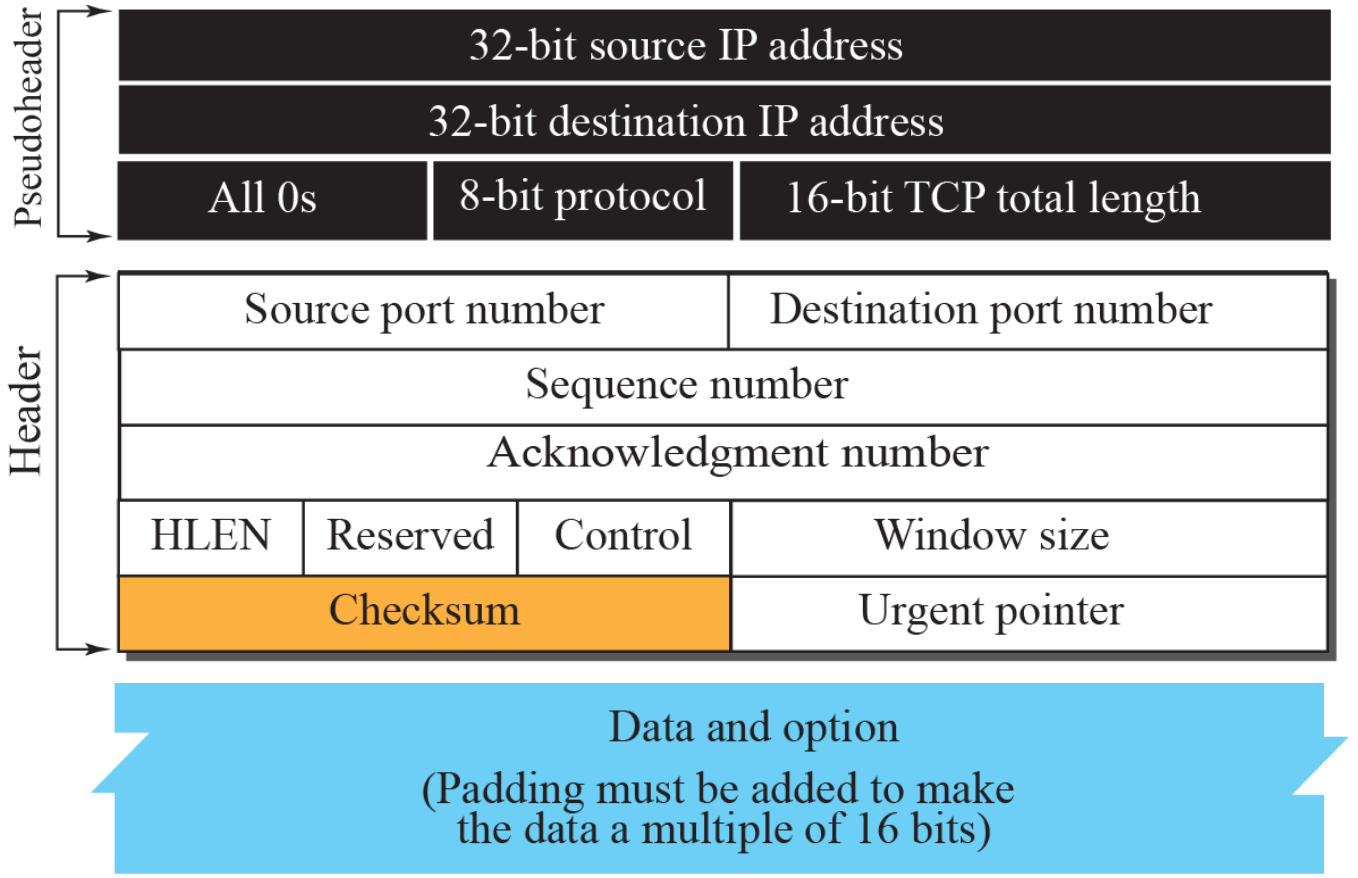


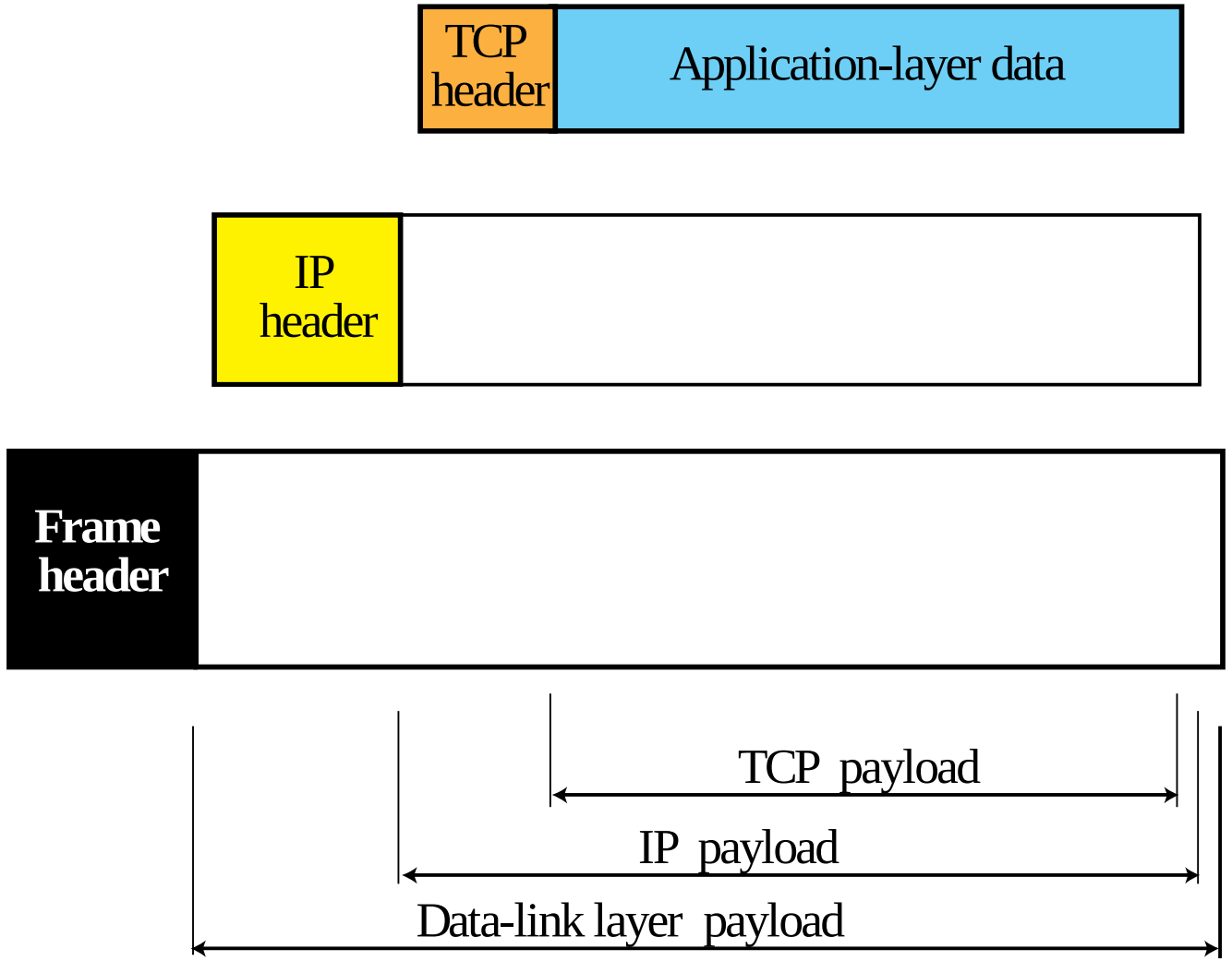
Figure 15.7 *Pseudoheader added to the TCP segment*



Note

The use of the checksum in TCP is mandatory.

Figure 15.8 *Encapsulation*



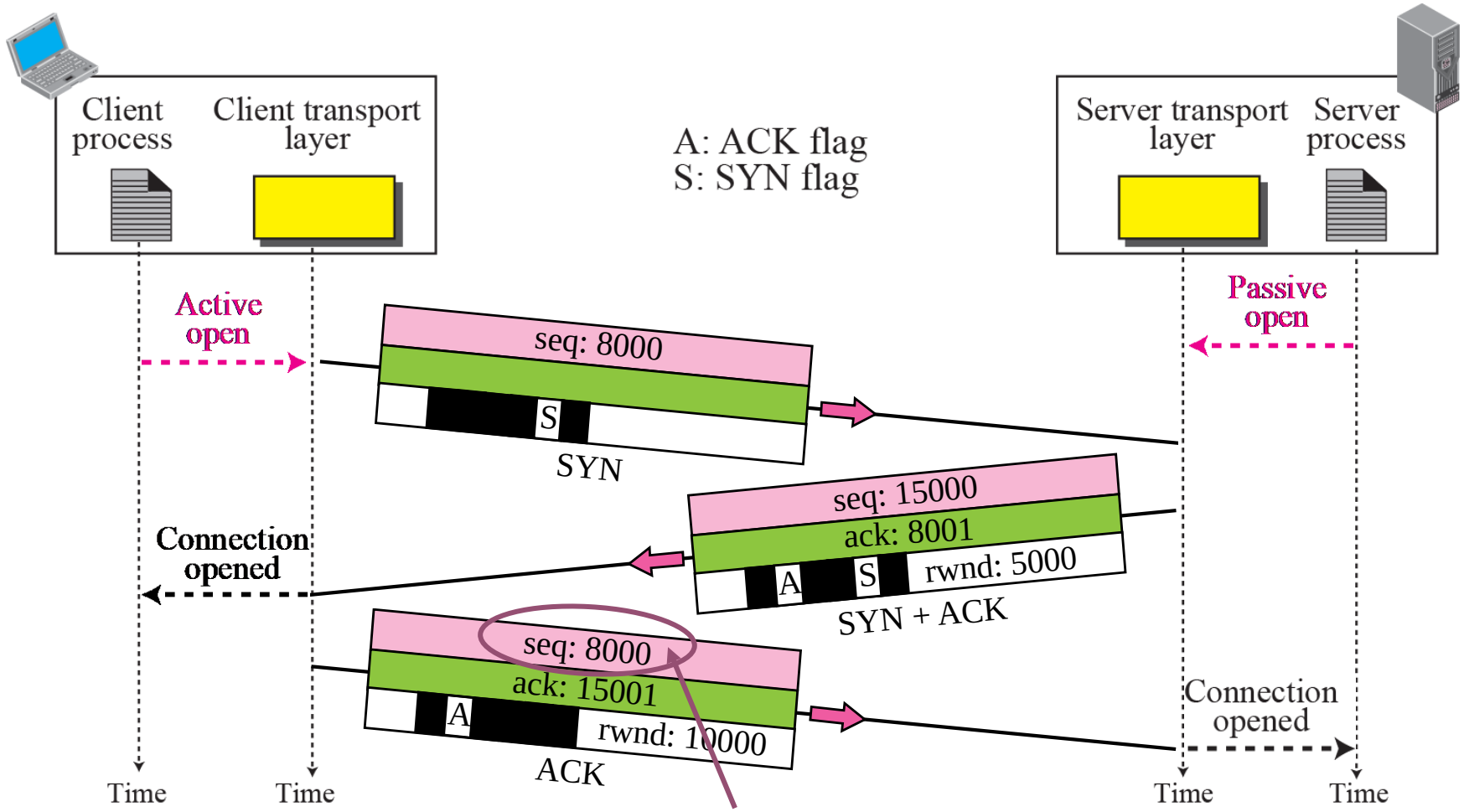
15-4 A TCP CONNECTION

TCP is connection-oriented. It establishes a virtual path between the source and destination. All of the segments belonging to a message are then sent over this virtual path. You may wonder how TCP, which uses the services of IP, a connectionless protocol, can be connection-oriented. The point is that a TCP connection is virtual, not physical. TCP operates at a higher level. TCP uses the services of IP to deliver individual segments to the receiver, but it controls the connection itself. If a segment is lost or corrupted, it is retransmitted.

Topics Discussed in the Section

- ✓ **Connection Establishment**
- ✓ **Data Transfer**
- ✓ **Connection Termination**
- ✓ **Connection Reset**

Figure 15.9 *Connection establishment using three-way handshake*



Note

A SYN segment cannot carry data, but it consumes one sequence number.

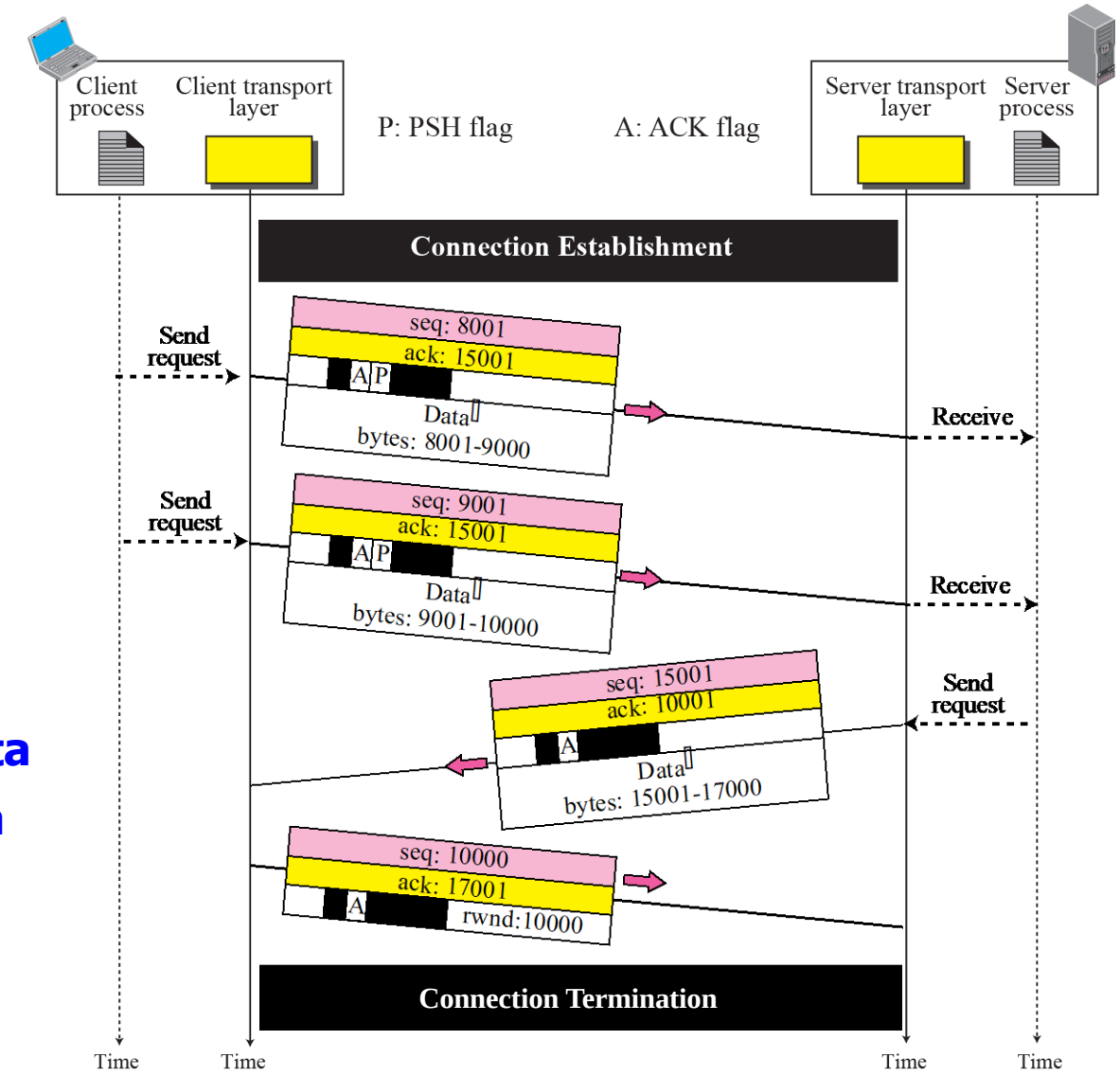
Note

A SYN + ACK segment cannot carry data, but does consume one sequence number.

Note

An ACK segment, if carrying no data, consumes no sequence number.

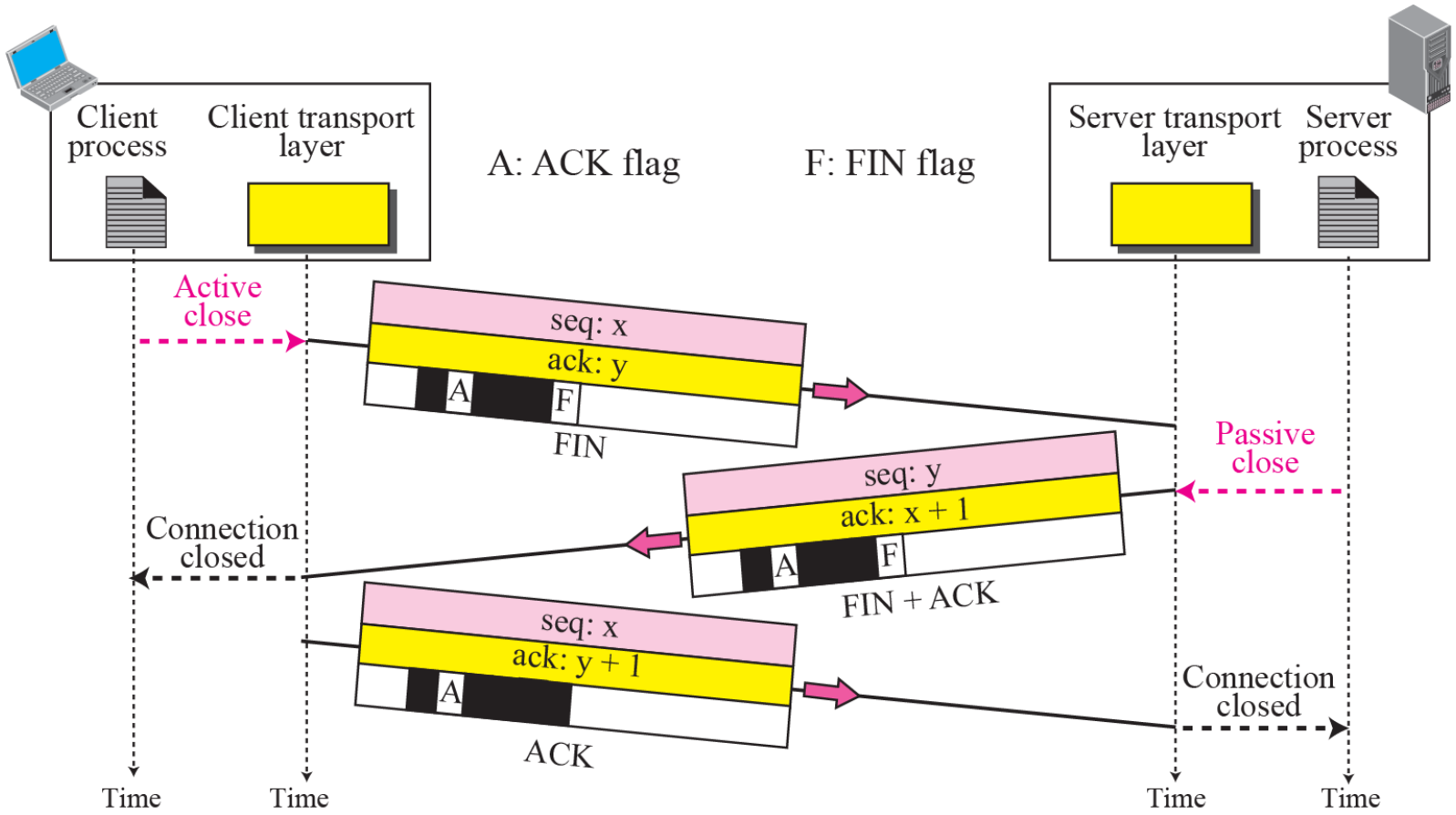
Figure 15.10 *Data Transfer*



Pushing data

Urgent data

Figure 15.11 *Connection termination using three-way handshake*



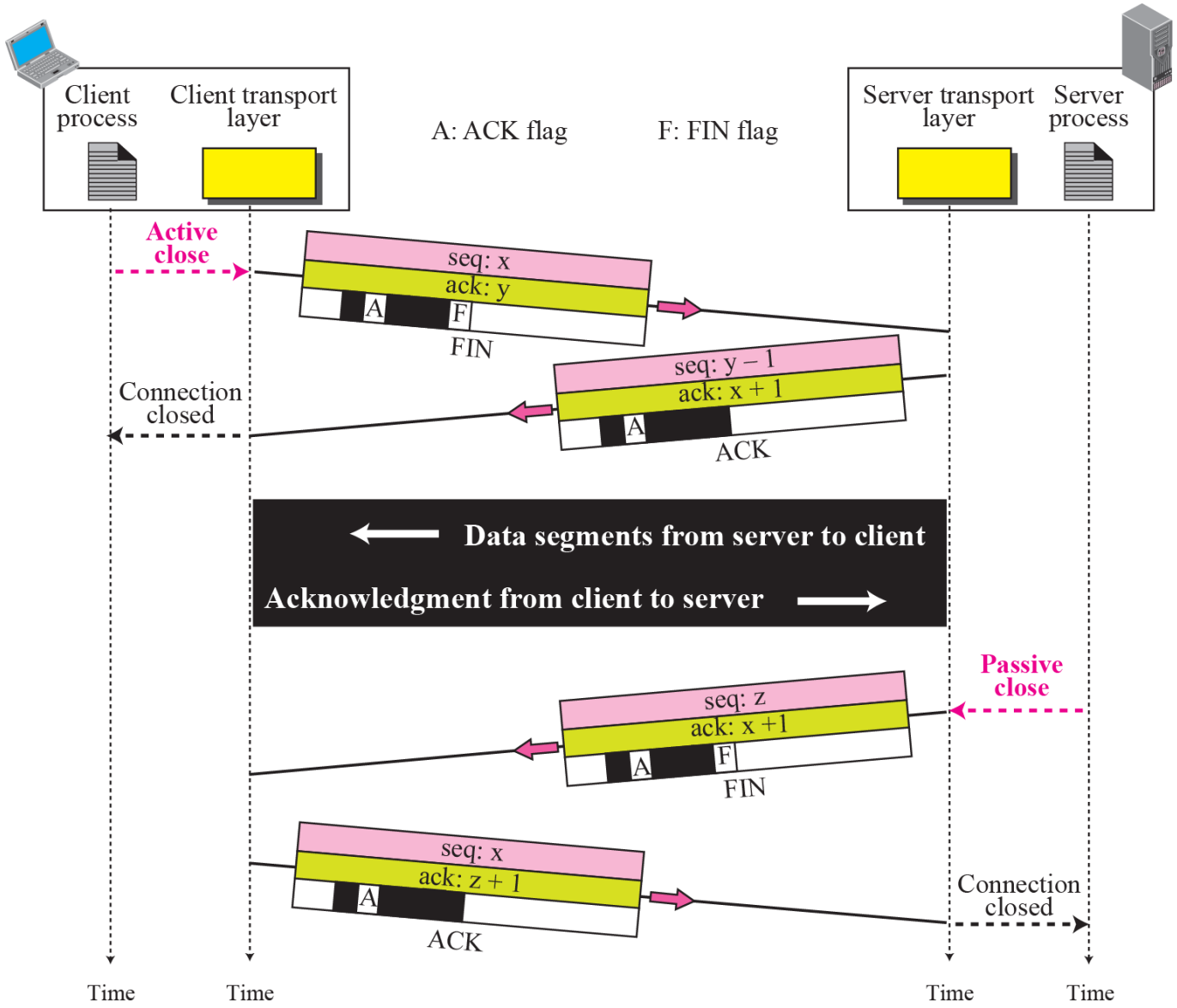
Note

The FIN segment consumes one sequence number if it does not carry data.

Note

The FIN + ACK segment consumes one sequence number if it does not carry data.

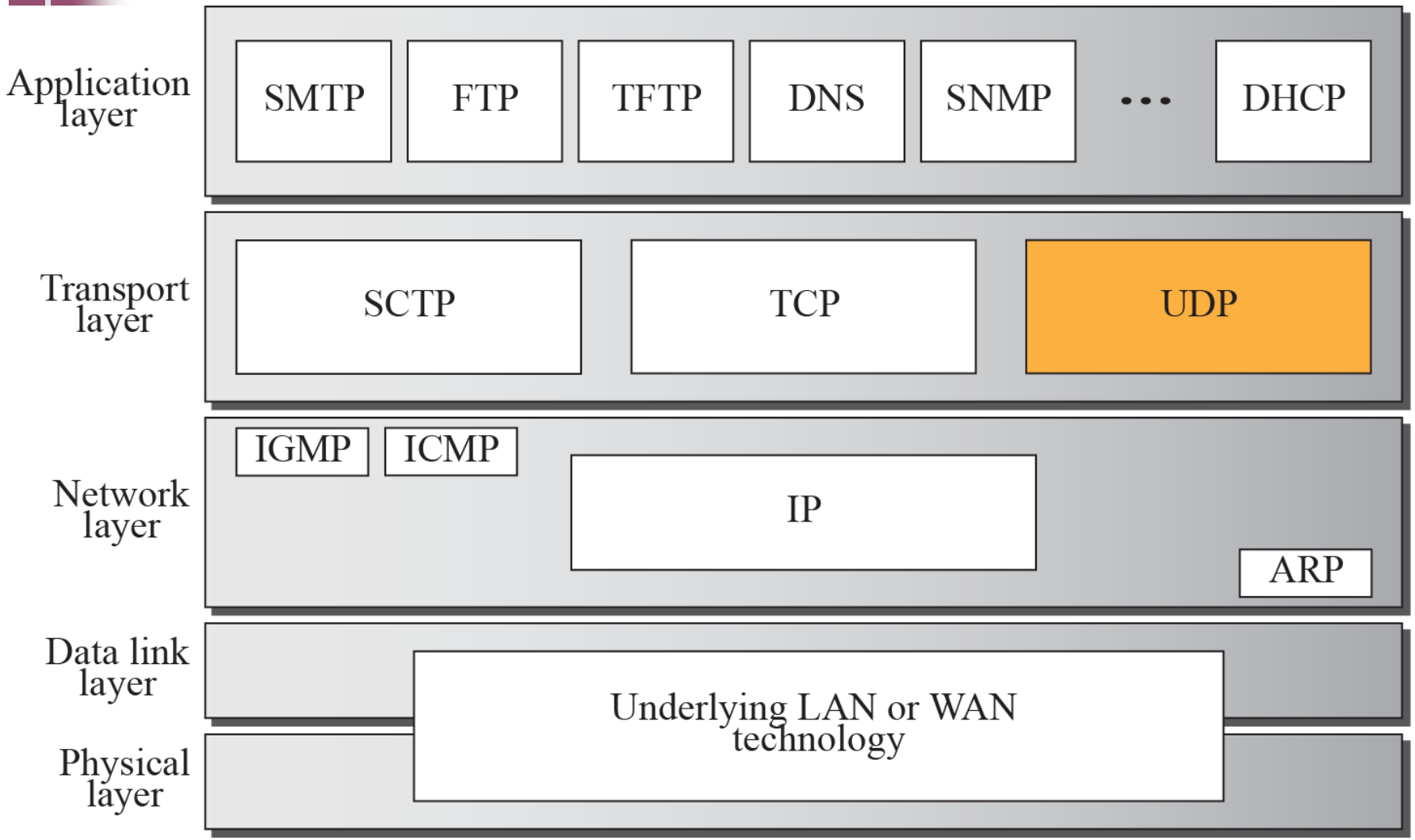
Figure 15.12 *Half-Close*



14-1 INTRODUCTION

Figure 14.1 shows the relationship of the User Datagram Protocol (UDP) to the other protocols and layers of the TCP/IP protocol suite: UDP is located between the application layer and the IP layer, and serves as the intermediary between the application programs and the network operations.

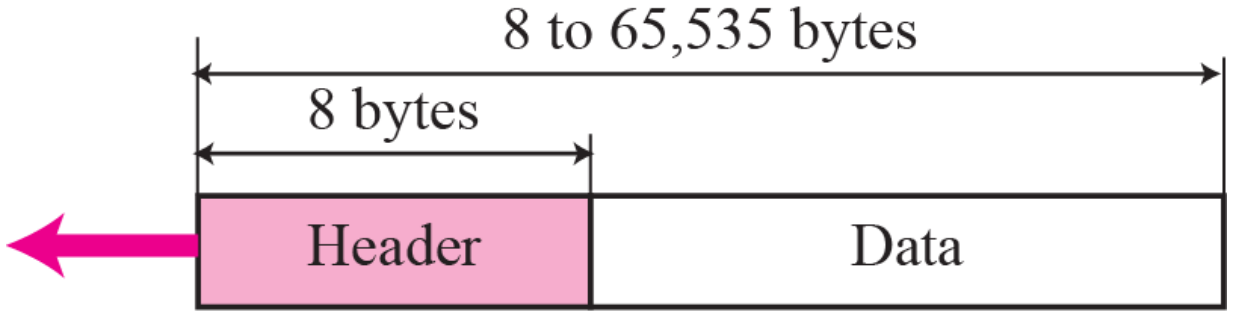
Figure 14.1 *Position of UDP in the TCP/IP protocol suite*



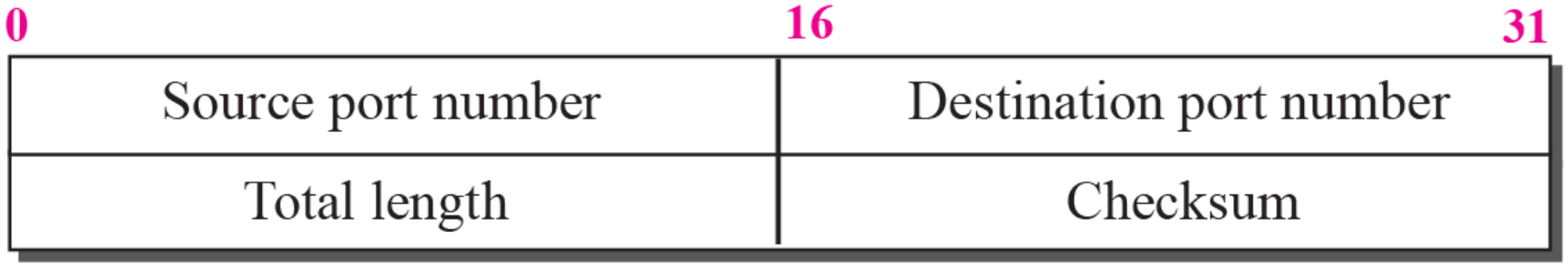
14-2 USER DATAGRAM

UDP packets, called user datagrams, have a fixed-size header of 8 bytes. Figure 14.2 shows the format of a user datagram.

Figure 14.2 *User datagram format*



a. UDP user datagram



b. Header format

Example 14.1

The following is a dump of a UDP header in hexadecimal format.

```
CB84000D001C001C
```

a. What is the source port number?
b. What is the destination port number?
c. What is the total length of the user datagram?
d. What is the length of the data?
e. Is the packet directed from a client to a server or vice versa?
f. What is the client process?

Example 14.1 *Continued*

Solution

a. The source port number is the first four hexadecimal digits $(CB84)_{16}$ or 52100.

b. The destination port number is the second four hexadecimal digits $(000D)_{16}$ or 13.

c. The third four hexadecimal digits $(001C)_{16}$ define the length of the whole UDP packet as 28 bytes.

d. The length of the data is the length of the whole packet minus the length of the header, or 28 – 8 = 20 bytes.

e. Since the destination port number is 13 (well-known port), the packet is from the client to the server.

f. The client process is the Daytime (see Table 14.1).

14-3 UDP Services

We discussed the general services provided by a transport layer protocol in Chapter 13. In this section, we discuss what portions of those general services are provided by UDP.

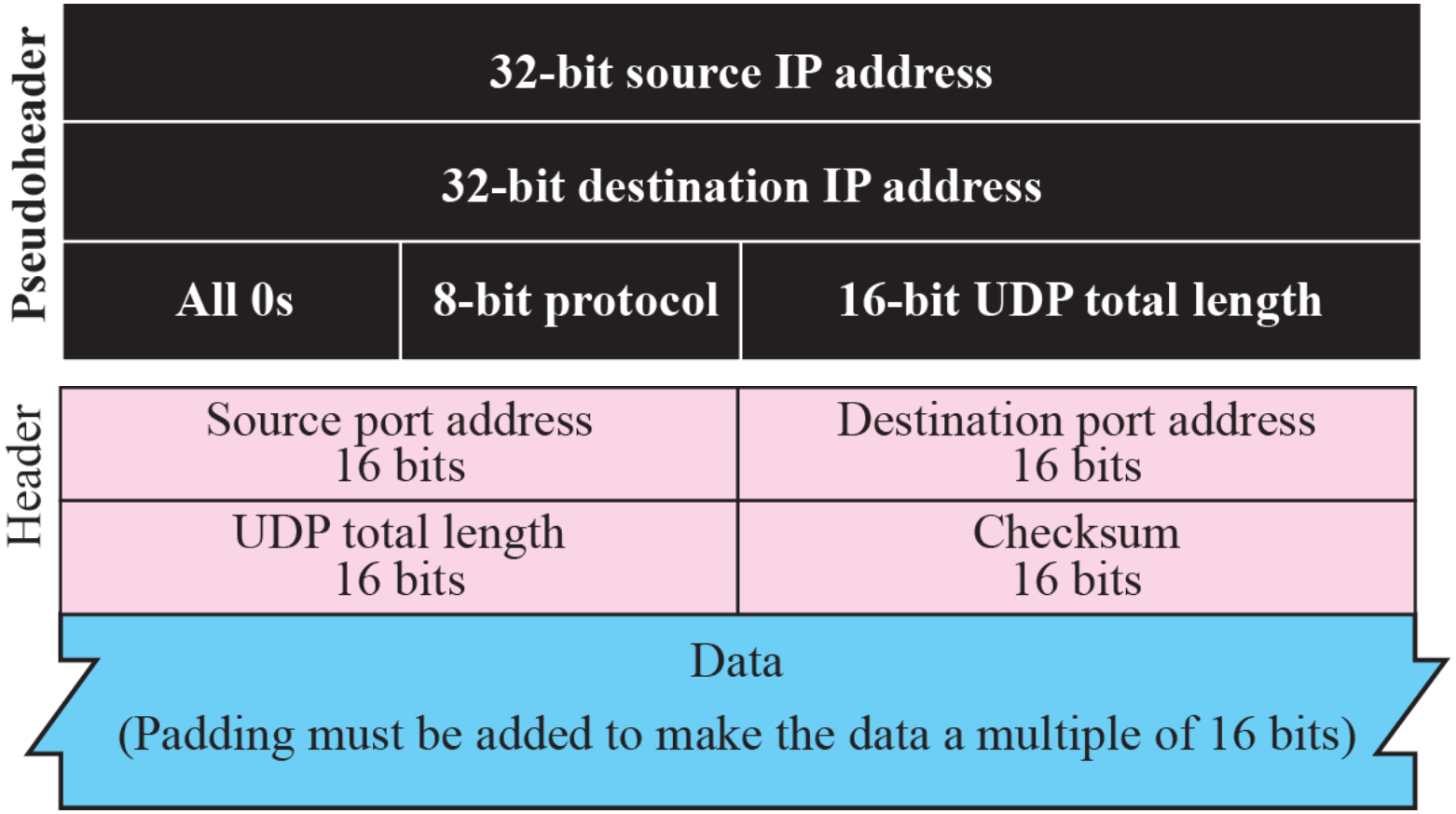
Topics Discussed in the Section

- ✓ **Process-to-Process Communication**
- ✓ **Connectionless Service**
- ✓ **Flow Control**
- ✓ **Error Control**
- ✓ **Congestion Control**
- ✓ **Encapsulation and Decapsulation**
- ✓ **Queuing**
- ✓ **Multiplexing and Demultiplexing**
- ✓ **Comparison between UDP and Generic Simple Protocol**

Table 14.1 *Well-known Ports used with UDP*

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
53	Domain	Domain Name Service (DNS)
67	Bootps	Server port to download bootstrap information
68	Bootpc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
111	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol (trap)

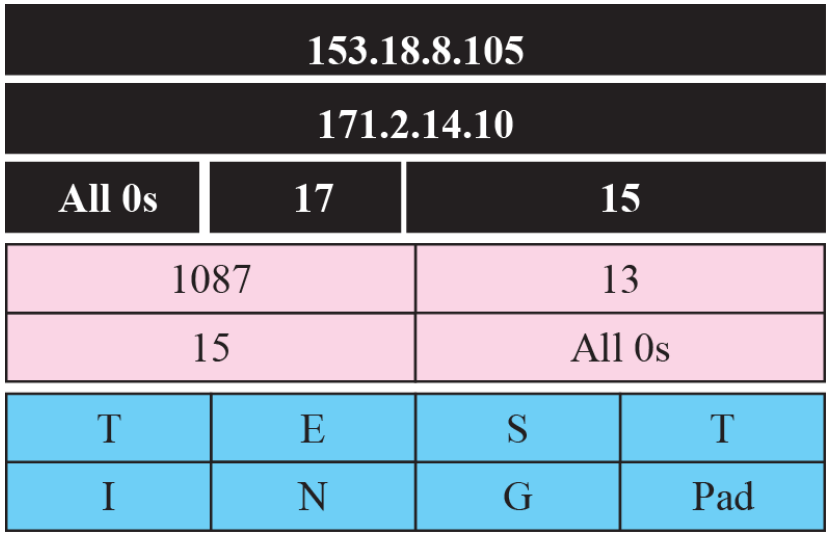
Figure 14.3 *Pseudoheader for checksum calculation*

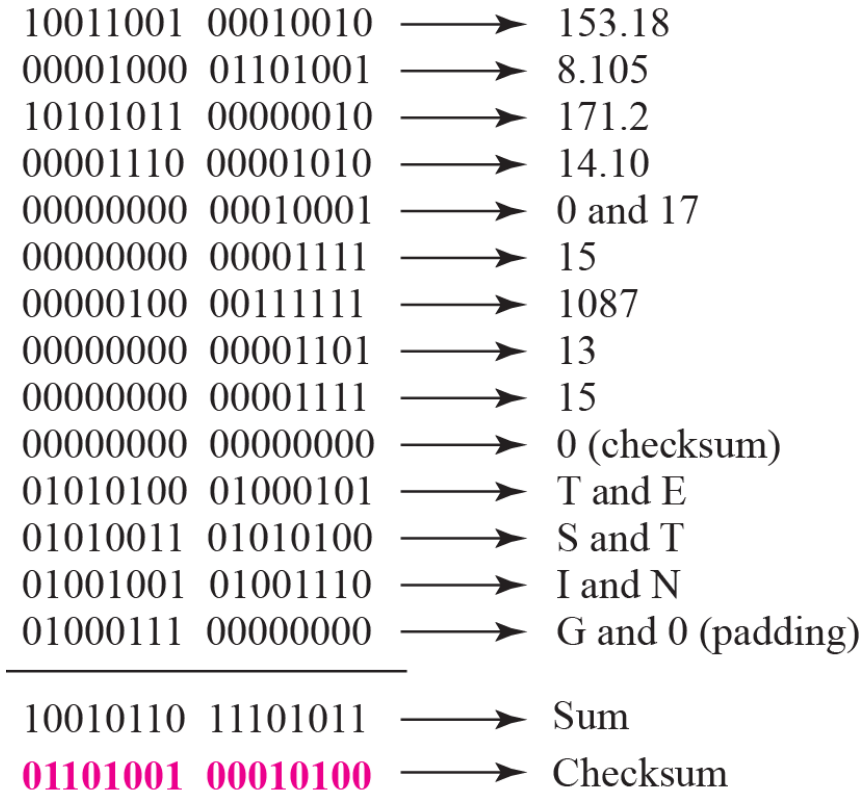


Example 14.2

Figure 14.4 shows the checksum calculation for a very small user datagram with only 7 bytes of data. Because the number of bytes of data is odd, padding is added for checksum calculation. The pseudoheader as well as the padding will be dropped when the user datagram is delivered to IP (see Appendix F).

Figure 14.4 *Checksum calculation for a simple UDP user datagram*





Example 14.3

What value is sent for the checksum in one of the following hypothetical situations?

a. The sender decides not to include the checksum.

b. The sender decides to include the checksum, but the value of the sum is all 1s.

c. The sender decides to include the checksum, but the value of the sum is all 0s.

Example 14.3 *Continued*

Solution

a. The value sent for the checksum field is all 0s to show that the checksum is not calculated.

b. When the sender complements the sum, the result is all 0s; the sender complements the result again before sending. The value sent for the checksum is all 1s. The second complement operation is needed to avoid confusion with the case in part a.

c. This situation never happens because it implies that the value of every term included in the calculation of the sum is all 0s, which is impossible; some fields in the pseudoheader have nonzero values (see Appendix D).

Figure 14.5 *Encapsulation and decapsulation*

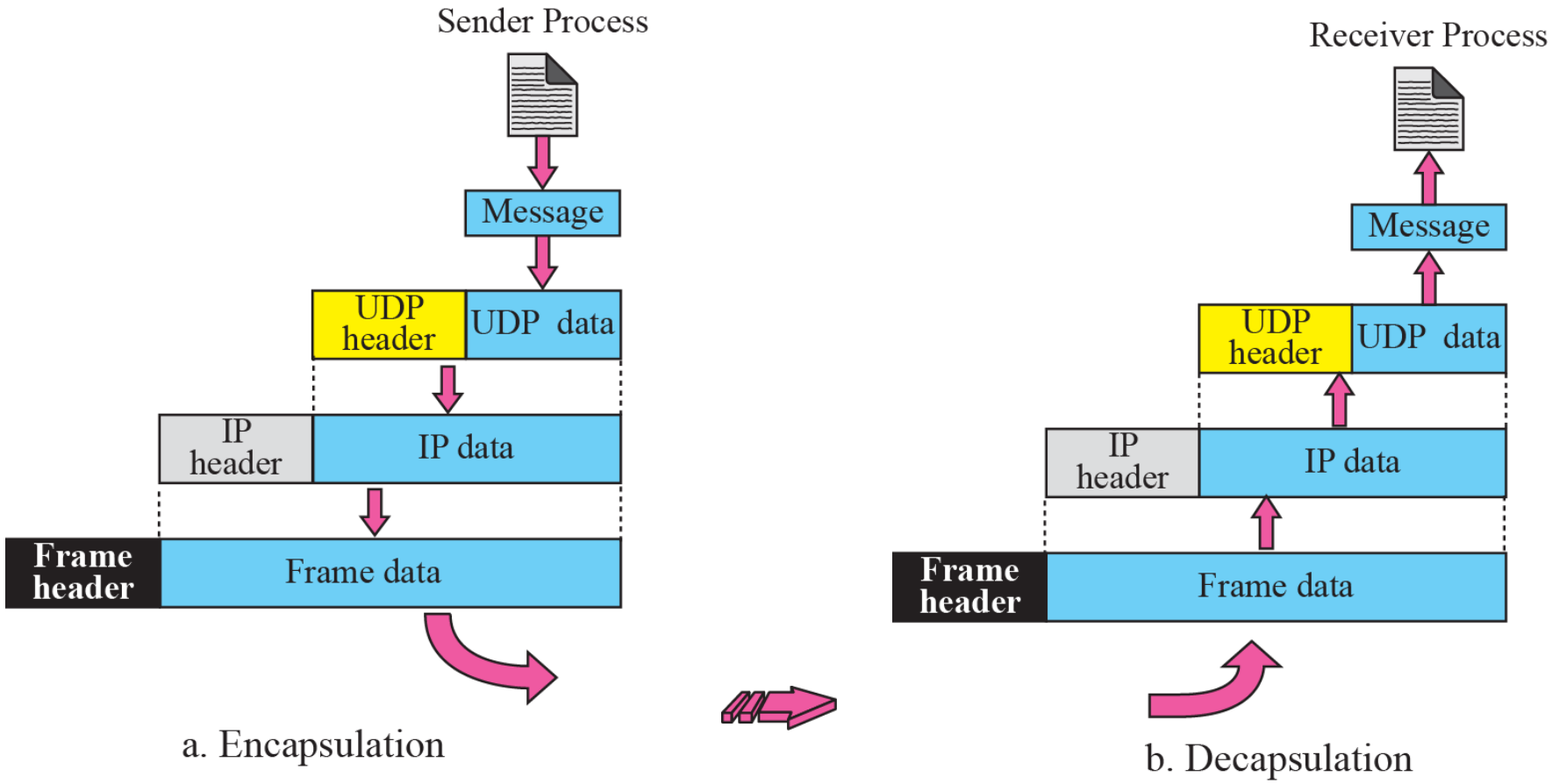


Figure 14.6 *Queues in UDP*

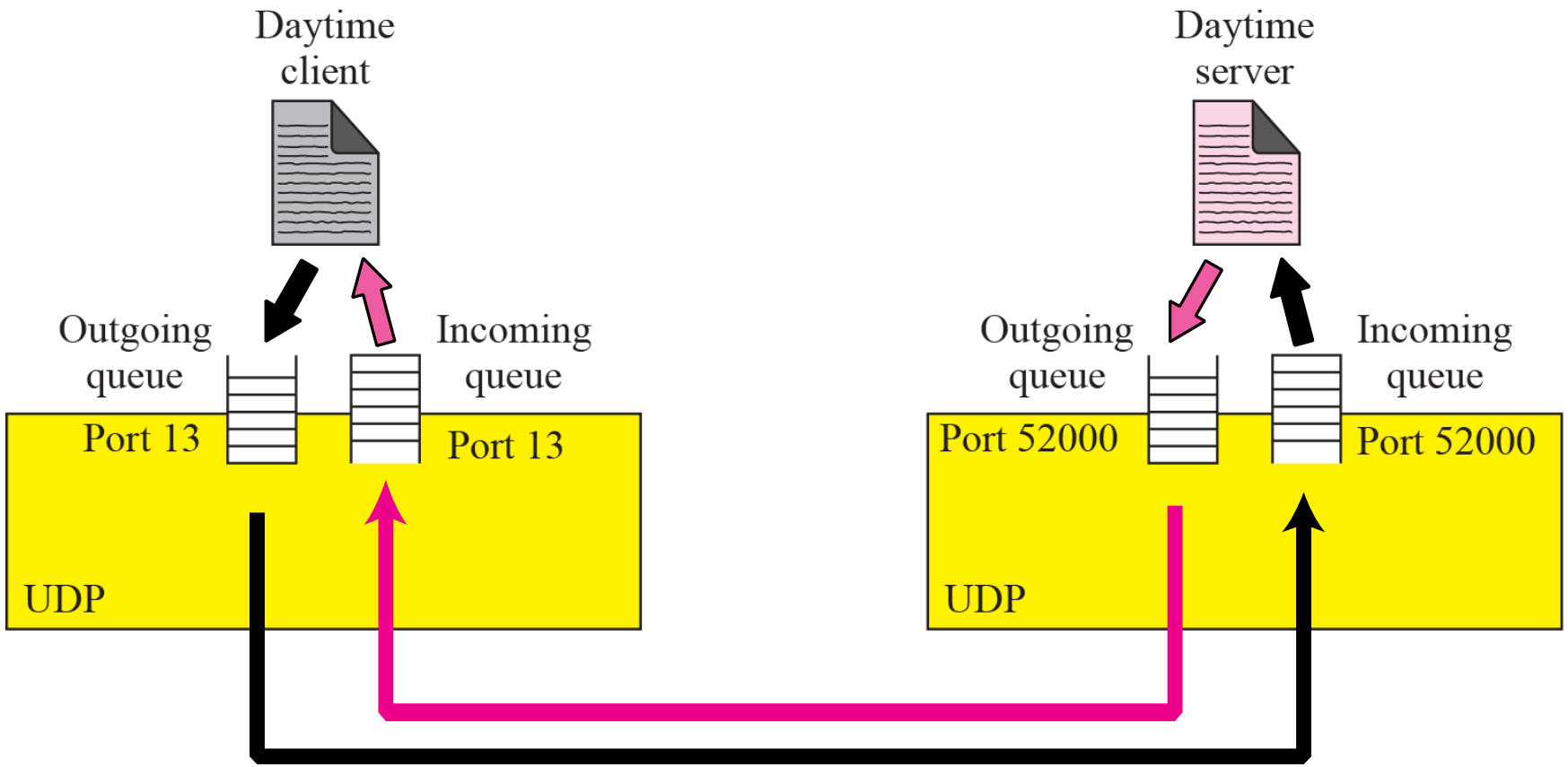
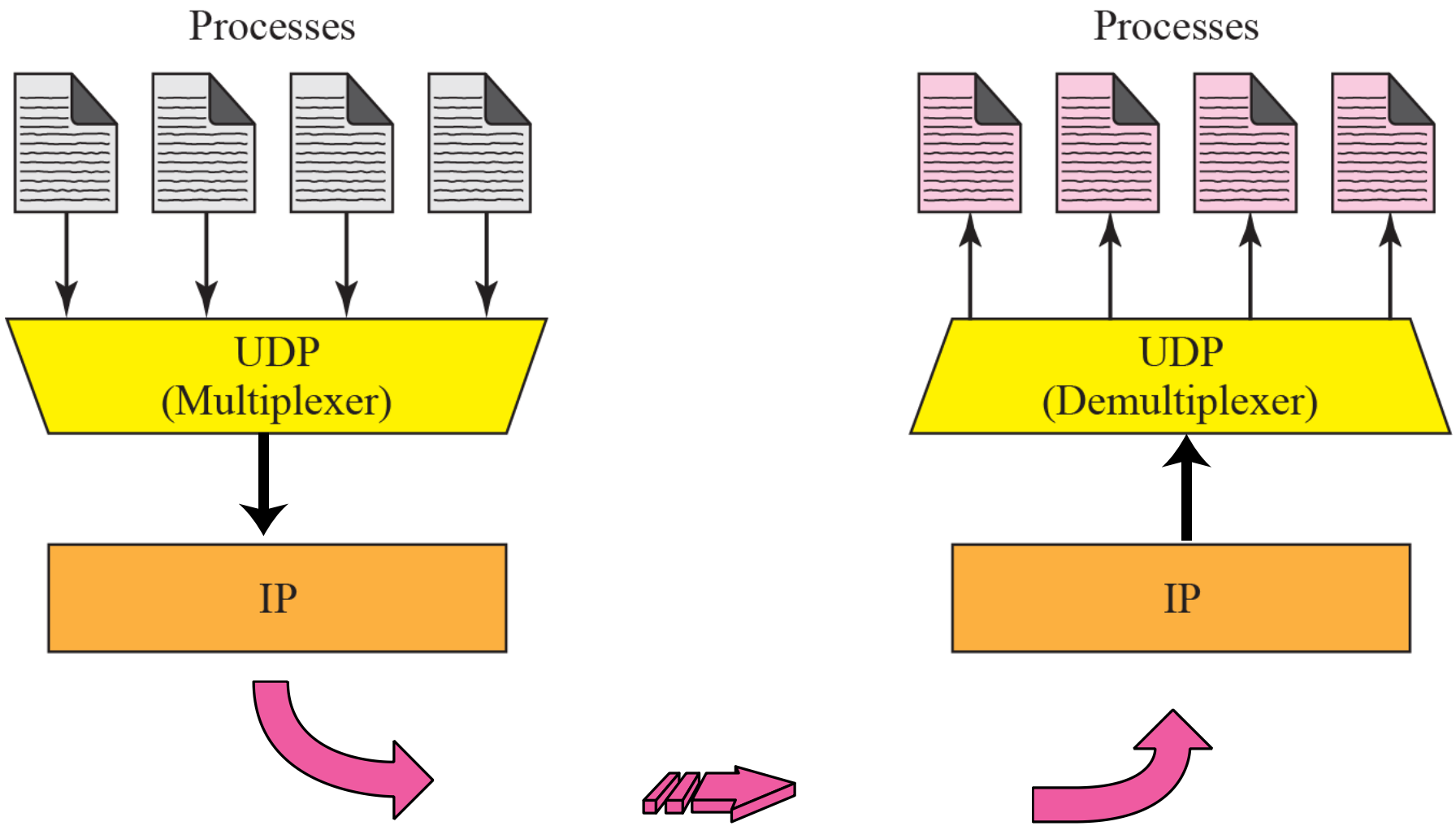


Figure 14.7 *Multiplexing and demultiplexing*



2. Multiplexing (1)

- The **network layer** provides communication between two hosts.
- The **transport layer** provides communication between two ***processes*** running on different hosts.
- A process is an instance of a program that is running on a host.
- There may be multiple processes communicating between two hosts – for example, there could be a FTP session and a Telnet session between the same two hosts.

Note

UDP is an example of the connectionless simple protocol we discussed in Chapter 13 with the exception of an optional checksum added to packets for error detection.

14-4 UDP APPLICATION

Although UDP meets almost none of the criteria we mentioned in Chapter 13 for a reliable transport-layer protocol, UDP is preferable for some applications. The reason is that some services may have some side effects that are either unacceptable or not preferable. An application designer needs sometimes to compromise to get the optimum.

Topics Discussed in the Section

- ✓ **UDP Features**
- ✓ **Typical Applications**

Example 14.4

A client-server application such as DNS (see Chapter 19) uses the services of UDP because a client needs to send a short request to a server and to receive a quick response from it. The request and response can each fit in one user datagram. Since only one message is exchanged in each direction, the connectionless feature is not an issue; the client or server does not worry that messages are delivered out of order.

Example 14.5

A client-server application such as SMTP (see Chapter 23), which is used in electronic mail, cannot use the services of UDP because a user can send a long e-mail message, which may include multimedia (images, audio, or video). If the application uses UDP and the message does not fit in one single user datagram, the message must be split by the application into different user datagrams. Here the connectionless service may create problems. The user datagrams may arrive and be delivered to the receiver application out of order. The receiver application may not be able to reorder the pieces. This means the connectionless service has a disadvantage for an application program that sends long messages.

Example 14.6

Assume we are downloading a very large text file from the Internet. We definitely need to use a transport layer that provides reliable service. We don't want part of the file to be missing or corrupted when we open the file. The delay created between the delivery of the parts are not an overriding concern for us; we wait until the whole file is composed before looking at it. In this case, UDP is not a suitable transport layer.

Example 14.7

Assume we are watching a real-time stream video on our computer. Such a program is considered a long file; it is divided into many small parts and broadcast in real time. The parts of the message are sent one after another. If the transport layer is supposed to resend a corrupted or lost frame, the synchronizing of the whole transmission may be lost. The viewer suddenly sees a blank screen and needs to wait until the second transmission arrives. This is not tolerable. However, if each small part of the screen is sent using one single user datagram, the receiving UDP can easily ignore the corrupted or lost packet and deliver the rest to the application program. That part of the screen is blank for a very short period of the time, which most viewers do not even notice. However, video cannot be viewed out of order, so streaming audio, video, and voice applications that run over UDP must reorder or drop frames that are out of sequence.

14-5 UDP PACKAGE

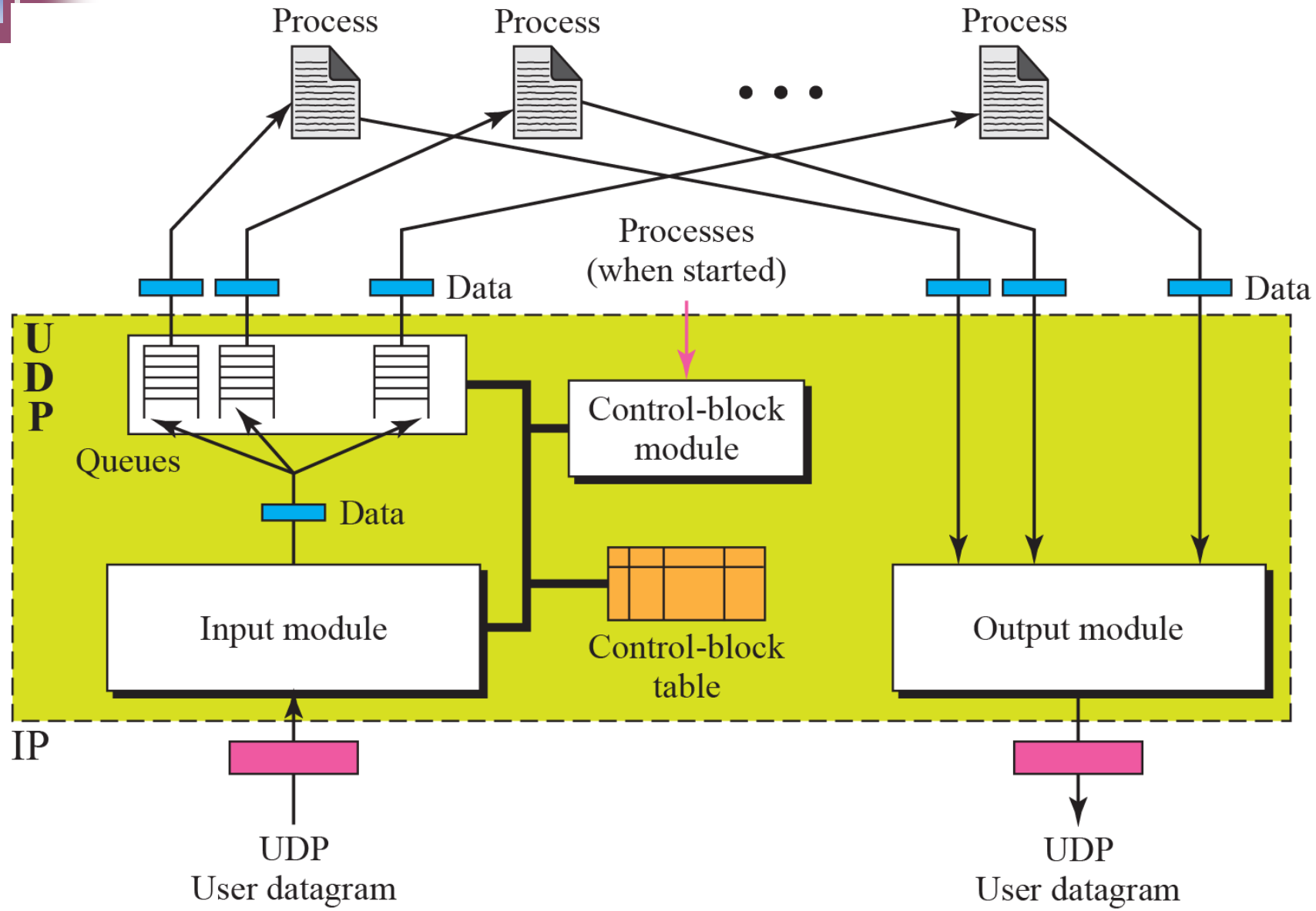
To show how UDP handles the sending and receiving of UDP packets, we present a simple version of the UDP package.

We can say that the UDP package involves five components: a control-block table, input queues, a control-block module, an input module, and an output module.

Topics Discussed in the Section

- ✓ **Control-Block Table**
- ✓ **Input Queues**
- ✓ **Control-Block Module**
- ✓ **Input Module**
- ✓ **Output Module**

Figure 14.8 *UDP design*



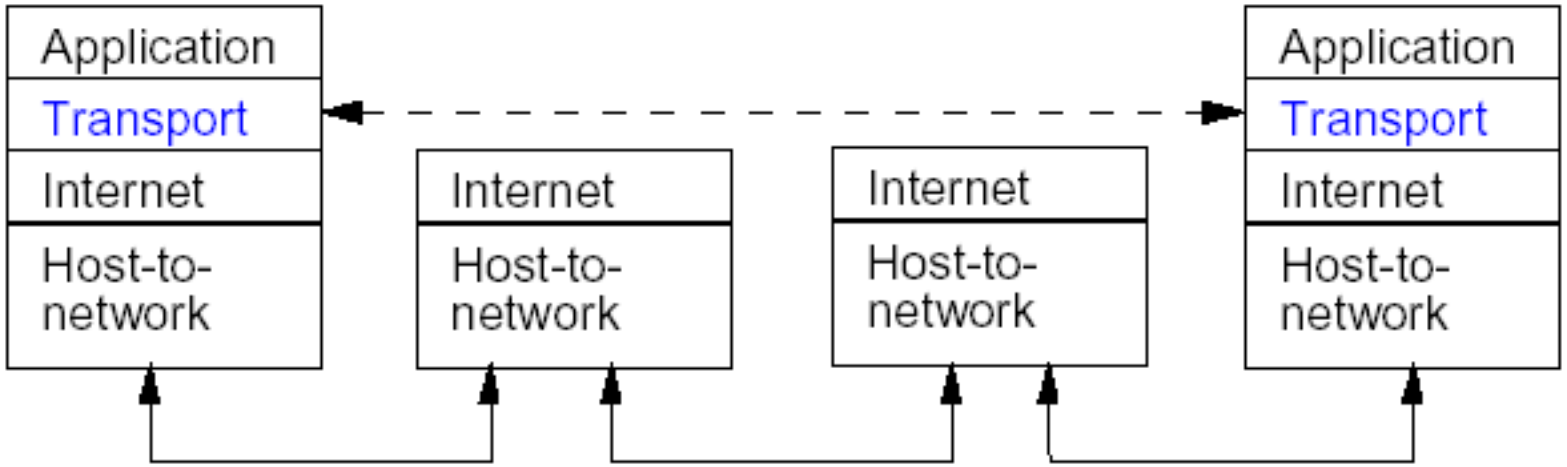
Summary

Part A: Introduction
Part B: Socket
Part C: TCP
Part D: UDP

Part A: Introduction

- The transport layer is an end-to-end layer – this means that nodes within the subnet do not participate in transport layer protocols – only the end hosts.

- As with other layers, transport layer protocols send data as a sequence of packets (segments).



3. Multiplexing (2)

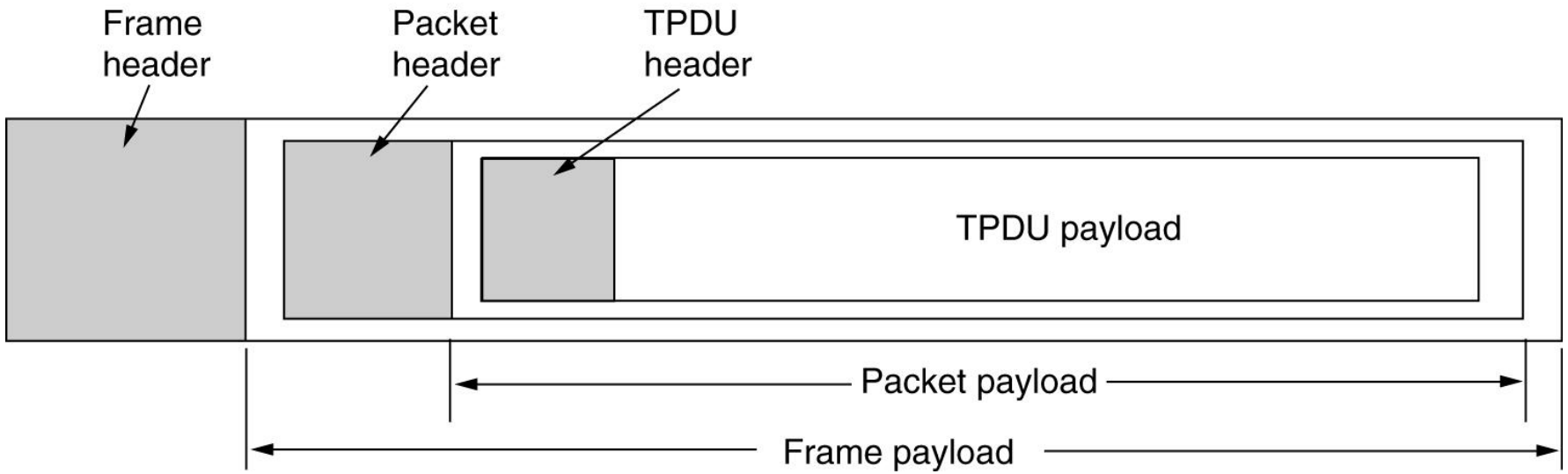
- The transport layer provides a way to multiplex / demultiplex communication between various processes.
- To provide multiplexing, the transport layer adds an address to each segment indicating the source and destination processes.
- Note these addresses need only be unique locally on a given host.
- In TCP/IP these transport layer addresses are called ***port-numbers***.

4. Multiplexing (3)

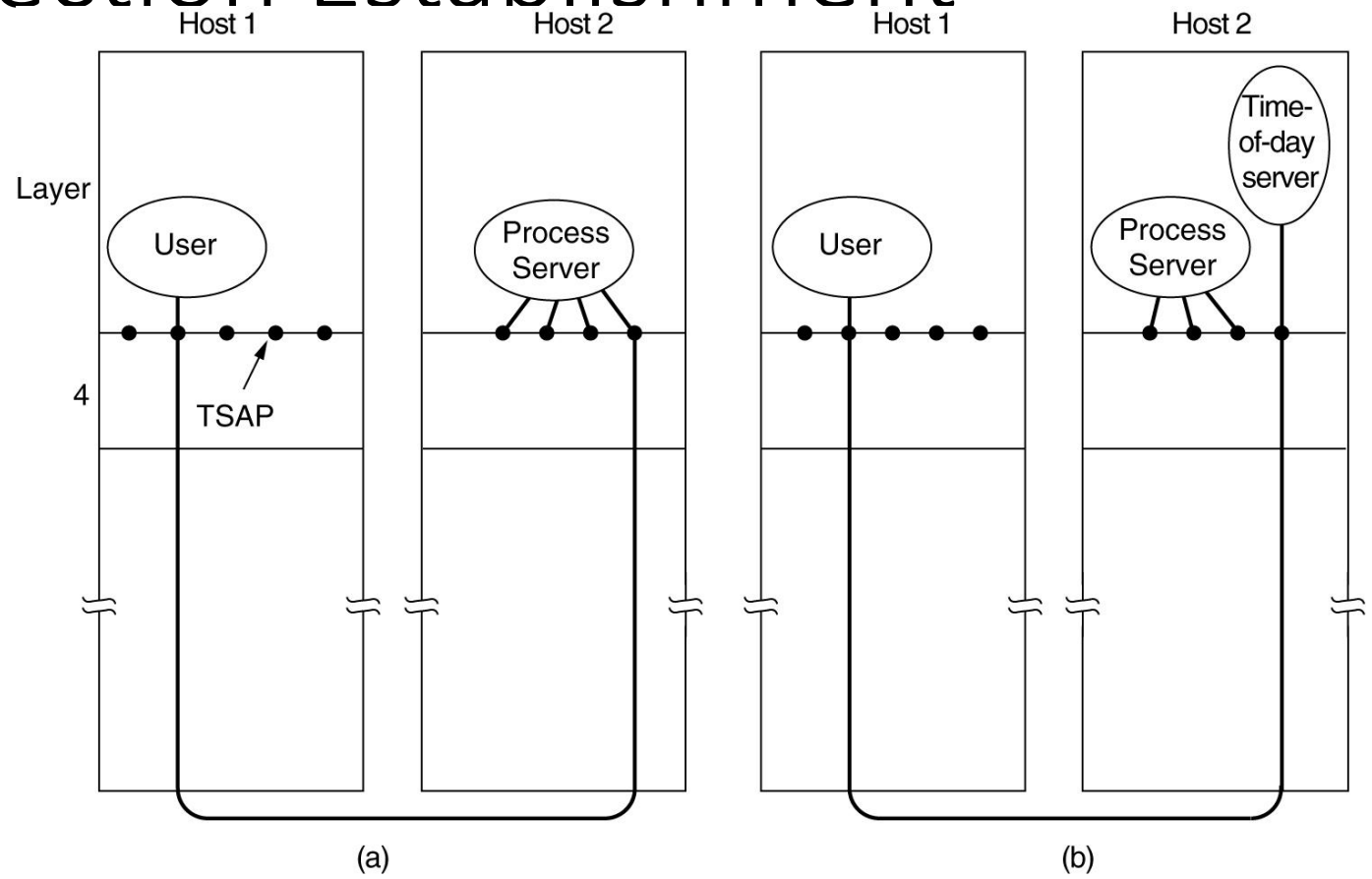
The 5-tuple: (sending port, sending IP address, destination port, destination IP address, transport layer protocol) uniquely identifies a process-to-process connection in the Internet.

5. TPDU

- Nesting of Transport Protocol Data Unit (TPDU), packets, and frames.



6. Connection Establishment



7. Transport Service Primitives

Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

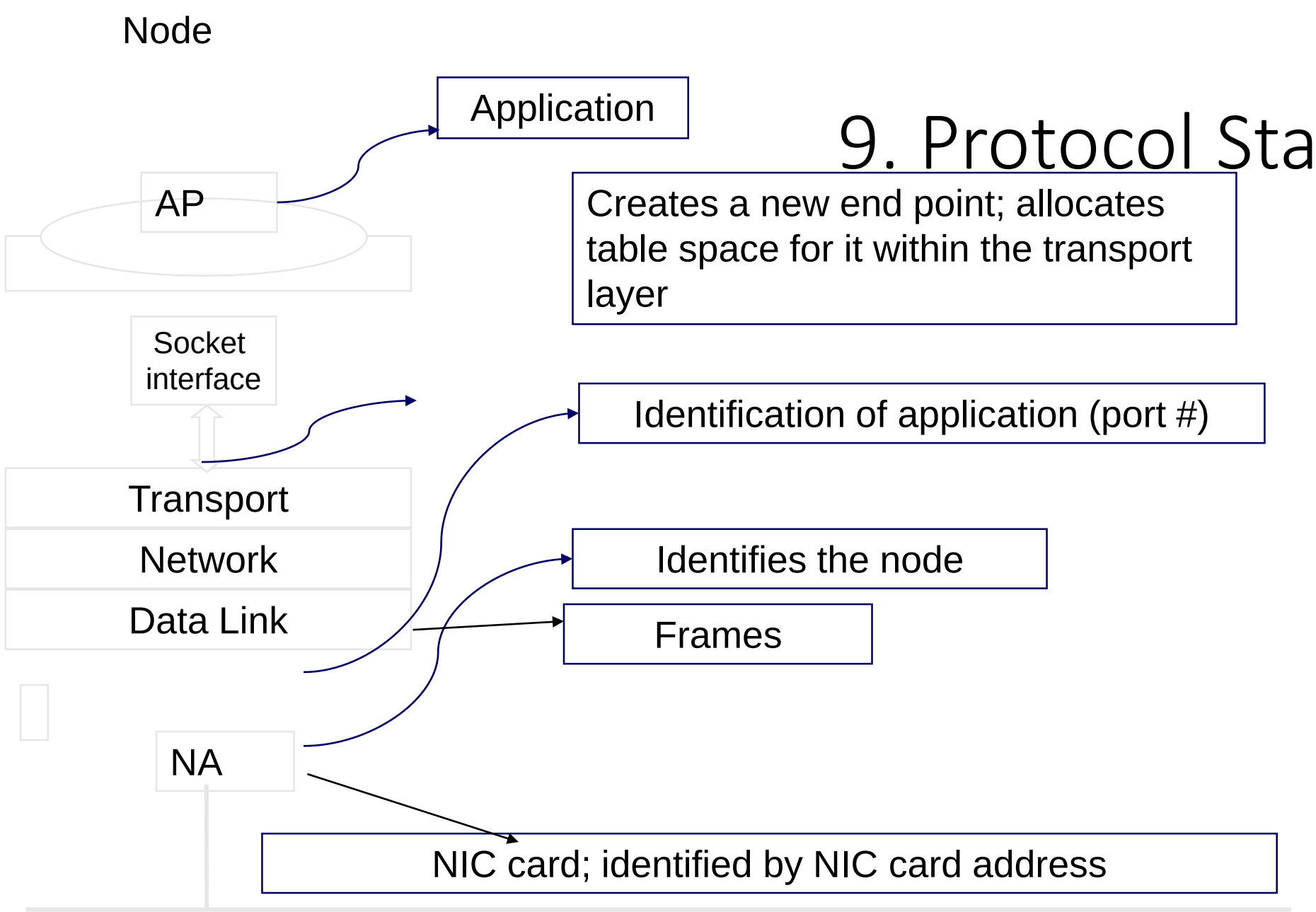
- The primitives for a simple transport service.

8. Transport Service Primitives

connum = LISTEN (local)
connum = CONNECT(local, remote)
status = SEND (connum, buffer, bytes)
status = RECEIVE(connum, buffer,bytes)
status = DISCONNECT(connum)

- The primitives for a simple transport service.

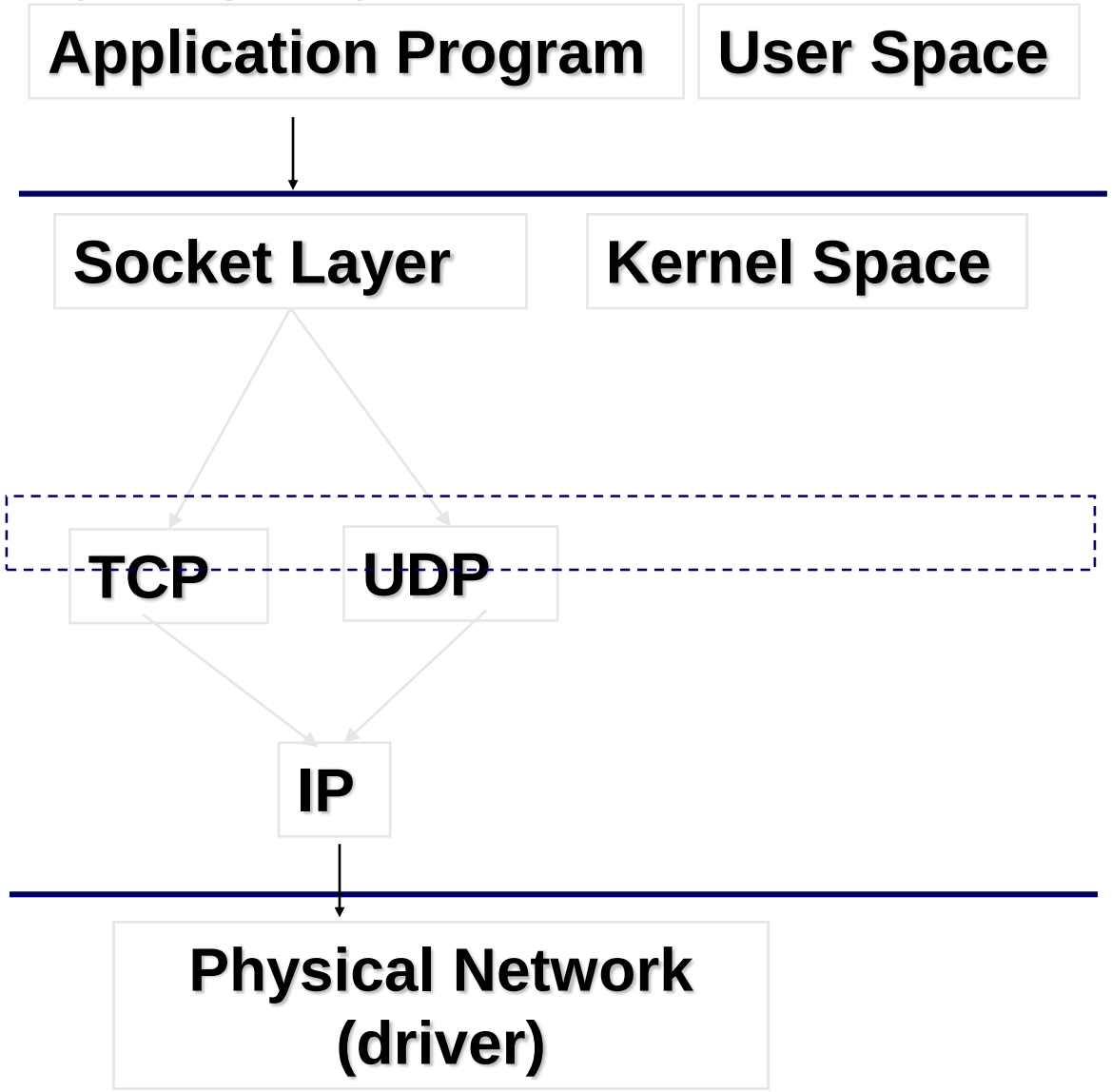
9. Protocol Stacks



PART B: SOCKET

- **Software interface designed to communicate between the user program and TCP/IP protocol stack**
- **Implemented by a set of library function calls**
- **Socket is a data structure inside the program**
- **Both client and server programs communicate via a pair of sockets**

2. Socket Framework



3. Socket Families

There are several significant socket domain families:

- ⇨Internet Domain Sockets (AF_INET)

  -implemented via IP addresses and port numbers

- ⇨Unix Domain Sockets (AF_UNIX)

  -implemented via filenames (think “named pipe”)

- ⇨Novell IPX (AF_IPX)

- ⇨AppleTalk DDS (AF_APPLETALK)

4. Type of Socket

- Stream (SOCK_STREAM) Uses TCP protocol. Connection-oriented service
- Datagram (SOCK_DGRAM) Uses UDP protocol. Connectionless service
- Raw (SOCK_RAW) Used for testing

5. Creating a Socket

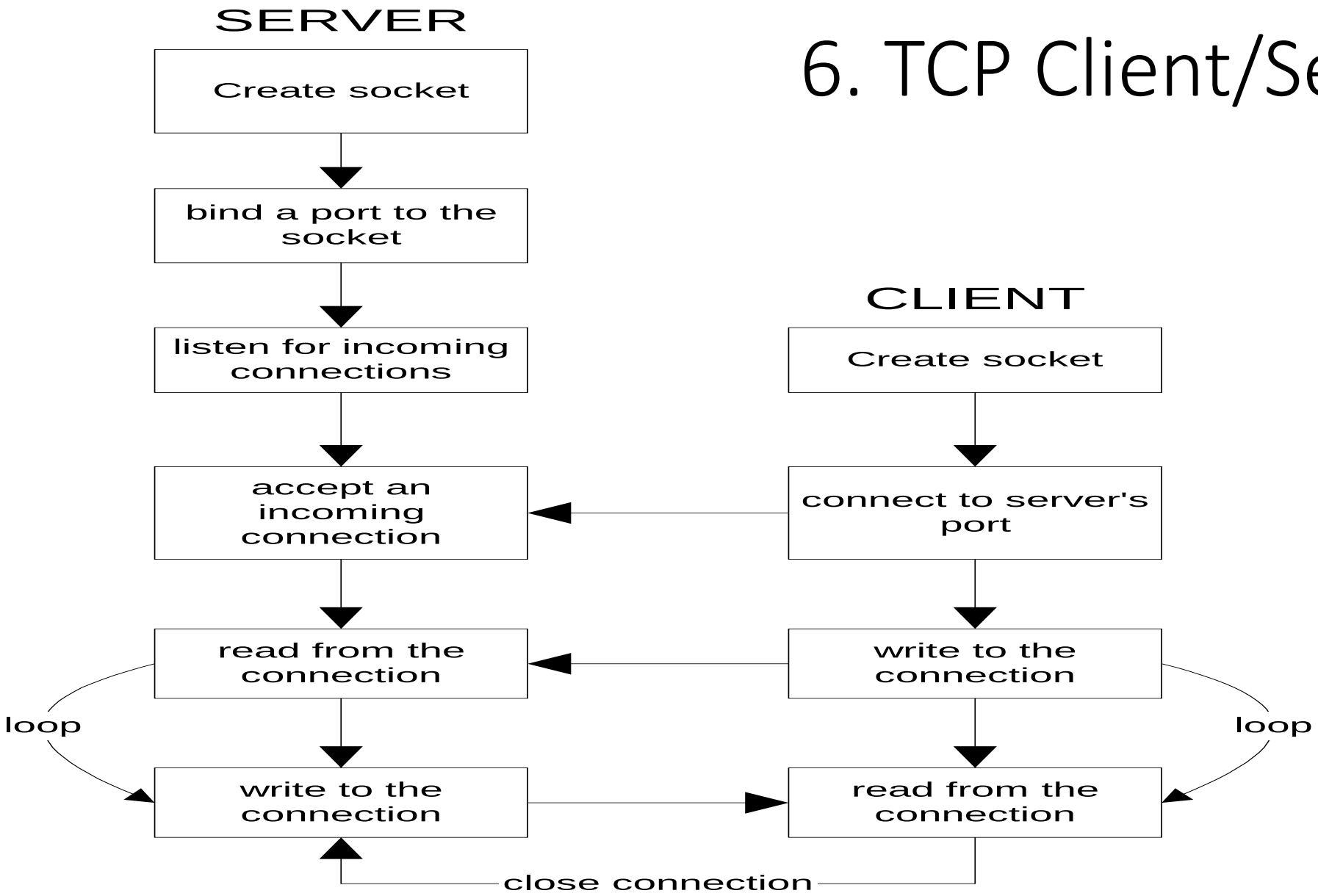
```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

- domain is one of the *Protocol Families* (PF_INET, PF_UNIX, etc.)

- type defines the communication protocol semantics, usually defines either:
  - SOCK_STREAM: connection-oriented stream (TCP)
  - SOCK_DGRAM: connectionless, unreliable (UDP)

- protocol specifies a particular protocol, just set this to 0 to accept the default

6. TCP Client/Server



7. TCP Server

- Sequence of calls

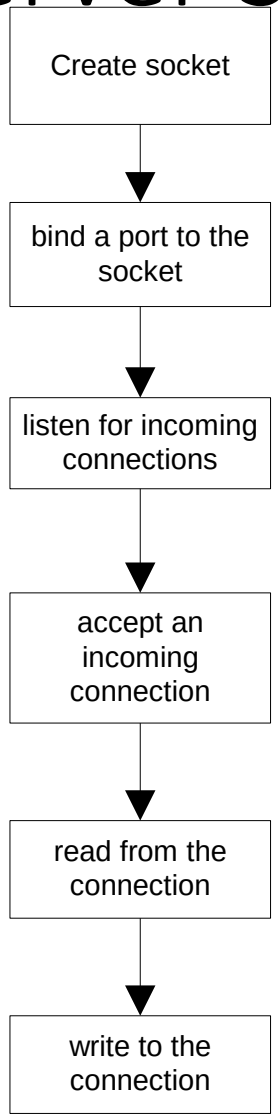
sock_init()	Create the socket
bind()	Register port with the system
listen()	Establish client connection
accept()	Accept client connection
read/write	read/write data
close()	shutdown

8. TCP Client

- Sequence of calls

sock_init ()	Create socket
connect()	Set up connection
write/read	write/read data
close()	Shutdown

9. Server Side Socket Details



int socket(int domain, int type, int protocol)
sockfd = socket(PF_INET, SOCK_STREAM, 0);

int bind(int sockfd, struct sockaddr *server_addr, socklen_t length)
bind(sockfd, &server, sizeof(server));

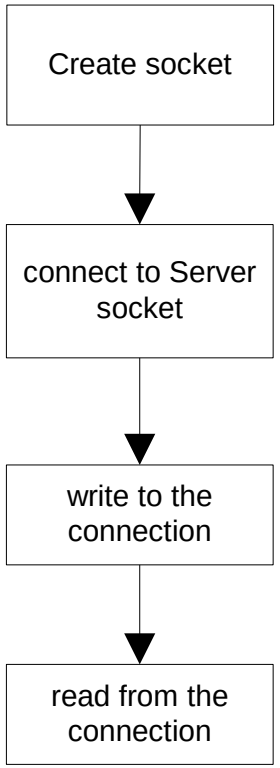
int listen(int sockfd, int num_queued_requests)
listen(sockfd, 5);

int accept(int sockfd, struct sockaddr *incoming_address, socklen_t length)
newfd = accept(sockfd, &client, sizeof(client)); /* BLOCKS */

int read(int sockfd, void * buffer, size_t buffer_size)
read(newfd, buffer, sizeof(buffer));

int write(int sockfd, void * buffer, size_t buffer_size)
write(newfd, buffer, sizeof(buffer));

10. Client Side Socket Details



int socket(int domain, int type, int protocol)
sockfd = socket(PF_INET, SOCK_STREAM, 0);

int connect(int sockfd, struct sockaddr *server_address, socklen_t length)
connect(sockfd, &server, sizeof(server));

int write(int sockfd, void * buffer, size_t buffer_size)
write(sockfd, buffer, sizeof(buffer));

int read(int sockfd, void * buffer, size_t buffer_size)
read(sockfd, buffer, sizeof(buffer));

11. UDP Clients and Servers

- Connectionless clients and servers create a socket using SOCK_DGRAM instead of SOCK_STREAM

- Connectionless servers do not call listen() or accept(), and *usually* do not call connect()

- Since connectionless communications lack a sustained connection, several methods are available that allow you to *specify a destination address with every call*:
  - sendto(sock, buffer, buflen, flags, to_addr, tolen);
  - recvfrom(sock, buffer, buflen, flags, from_addr, fromlen);

12. UDP Server

- Sequence of calls

sock_init()	Create socket
bind()	Register port with the system
recfrom/sendto	Receive/send data
close()	Shutdown

13. UDP Client

- Sequence of calls

socket_init()	Create socket
sendto/recfrom	send/receive data
close()	Shutdown

14. Socket Programming Example: Internet File Server

- Client code using sockets.

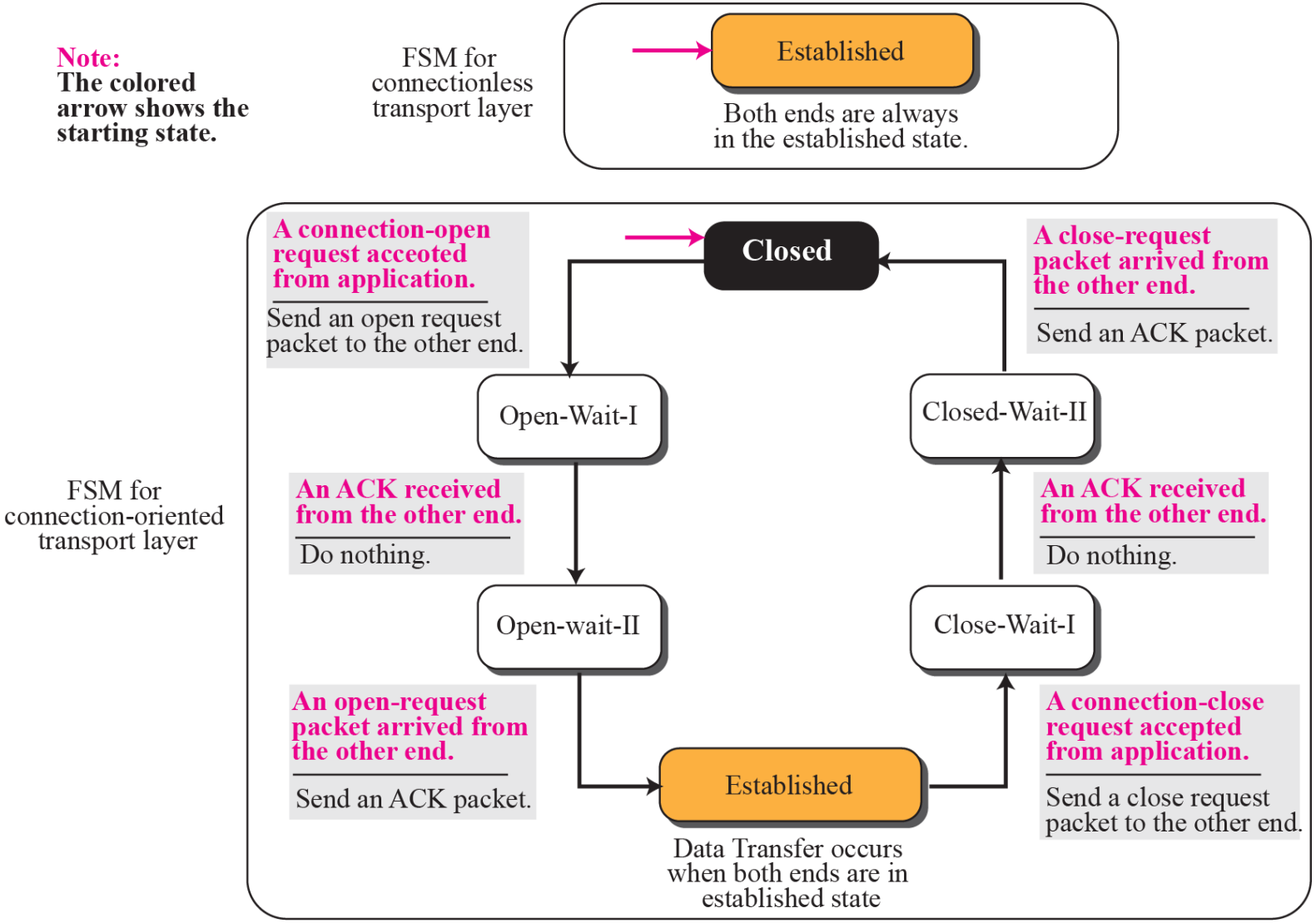
```
/* This page contains a client program that can request a file from the server program
 * on the next page. The server responds by sending the whole file.
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345                /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096                    /* block transfer size */

int main(int argc, char **argv)
{
  int c, s, bytes;
  char buf[BUF_SIZE];                    /* buffer for incoming file */
  struct hostent *h;                     /* info about server */
  struct sockaddr_in channel;            /* holds IP address */

  if (argc != 3) fatal("Usage: client server-name file-name");
  h = gethostbyname(argv[1]);            /* look up host's IP address */
  if (!h) fatal("gethostbyname failed");

  s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
  if (s <0) fatal("socket");
  memset(&channel, 0, sizeof(channel));
  channel.sin_family= AF_INET;
  memcpy(&channel.sin_addr.s_addr, h->h_addr, h->h_length);
  channel.sin_port= htons(SERVER_PORT);

  c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
  if (c < 0) fatal("connect failed");

  /* Connection is now established. Send file name including 0 byte at end. */
  write(s, argv[2], strlen(argv[2])+1);

  /* Go get the file and write it to standard output. */
  while (1) {
       bytes = read(s, buf, BUF_SIZE);   /* read from socket */
       if (bytes <= 0) exit(0);          /* check for end of file */
       write(1, buf, bytes);             /* write to standard output */
  }
}

fatal(char *string)
{
  printf("%s\n", string);
  exit(1);
}
```

15. Socket Programming Example: Internet File Server (2)

- Server code using sockets.

```
#include <sys/types.h>                  /* This is the server code */
#include <sys/fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define SERVER_PORT 12345               /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096                   /* block transfer size */
#define QUEUE_SIZE 10
int main(int argc, char *argv[])
{
  int s, b, l, fd, sa, bytes, on = 1;
  char buf[BUF_SIZE];                   /* buffer for outgoing file */
  struct sockaddr_in channel;           /* hold's IP address */

  /* Build address structure to bind to socket. */
  memset(&channel, 0, sizeof(channel));  /* zero channel */
  channel.sin_family = AF_INET;
  channel.sin_addr.s_addr = htonl(INADDR_ANY);
  channel.sin_port = htons(SERVER_PORT);

  /* Passive open. Wait for connection. */
  s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);    /* create socket */
  if (s < 0) fatal("socket failed");
  setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *) &on, sizeof(on));

  b = bind(s, (struct sockaddr *) &channel, sizeof(channel));
  if (b < 0) fatal("bind failed");

  l = listen(s, QUEUE_SIZE);            /* specify queue size */
  if (l < 0) fatal("listen failed");

  /* Socket is now set up and bound. Wait for connection and process it. */
  while (1) {
      sa = accept(s, 0, 0);             /* block for connection request */
      if (sa < 0) fatal("accept failed");

      read(sa, buf, BUF_SIZE);          /* read file name from socket */

      /* Get and return the file. */
      fd = open(buf, O_RDONLY);         /* open the file to be sent back */
      if (fd < 0) fatal("open failed");

      while (1) {
          bytes = read(fd, buf, BUF_SIZE); /* read from file */
          if (bytes <= 0) break;        /* check for end of file */
          write(sa, buf, bytes);        /* write bytes to socket */
      }
      close(fd);                        /* close file */
      close(sa);                        /* close connection */
  }
}
```

Figure 13.15 *Connectionless and connection-oriented services as FSMs*



Example 13.2

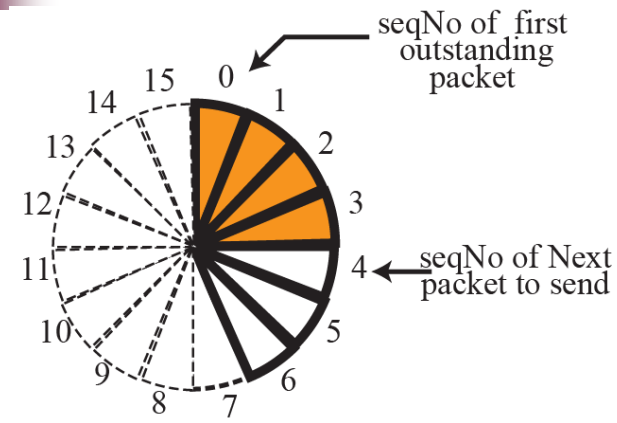
The above discussion requires that the consumers communicate with the producers in two occasions: when the buffer is full and when there are vacancies. If the two parties use a buffer of only one slot, the communication can be easier. Assume that each transport layer uses one single memory location to hold a packet. When this single slot in the sending transport layer is empty, the sending transport layer sends a note to the application layer to send its next chunk; when this single slot in the receiving transport layer is empty, it sends an acknowledgment to the sending transport layer to send its next packet. As we will see later, this type of flow control, using a single-slot buffer at the sender and the receiver, is inefficient.

Figure 13.10 *Error control at the transport layer*

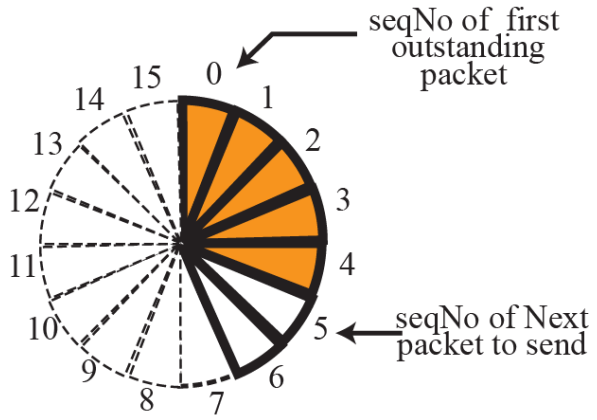
Note

For error control, the sequence numbers are modulo 2^m, where m is the size of the sequence number field in bits.

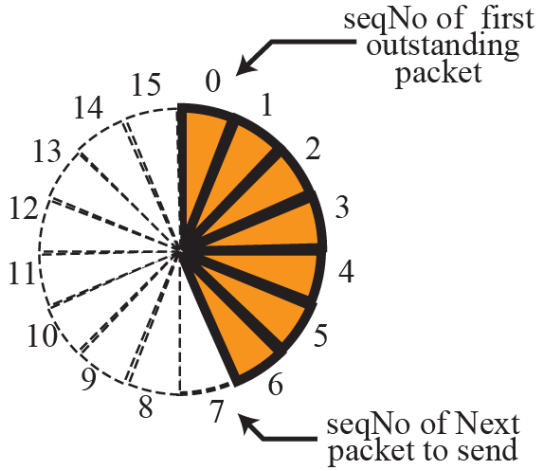
Figure 13.11 *Sliding window in circular format*



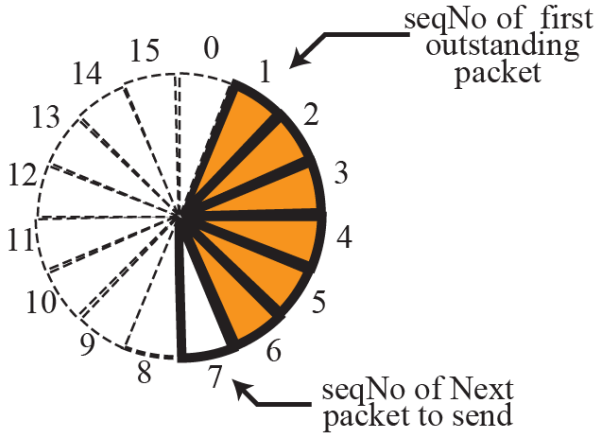
a. Four packets have been sent



b. Five packets have been sent



c. Seven packets have been sent window is full



d. Packet 0 has been acknowledged, window slides

Figure 13.12 *Sliding window in linear format*

a. Four packets have been sent

b. Five packets have been sent

c. Seven packets have been sent
window is full

d. Packet 0 have been acknowledged
and window slid

Figure 13.13 *Connectionless service*

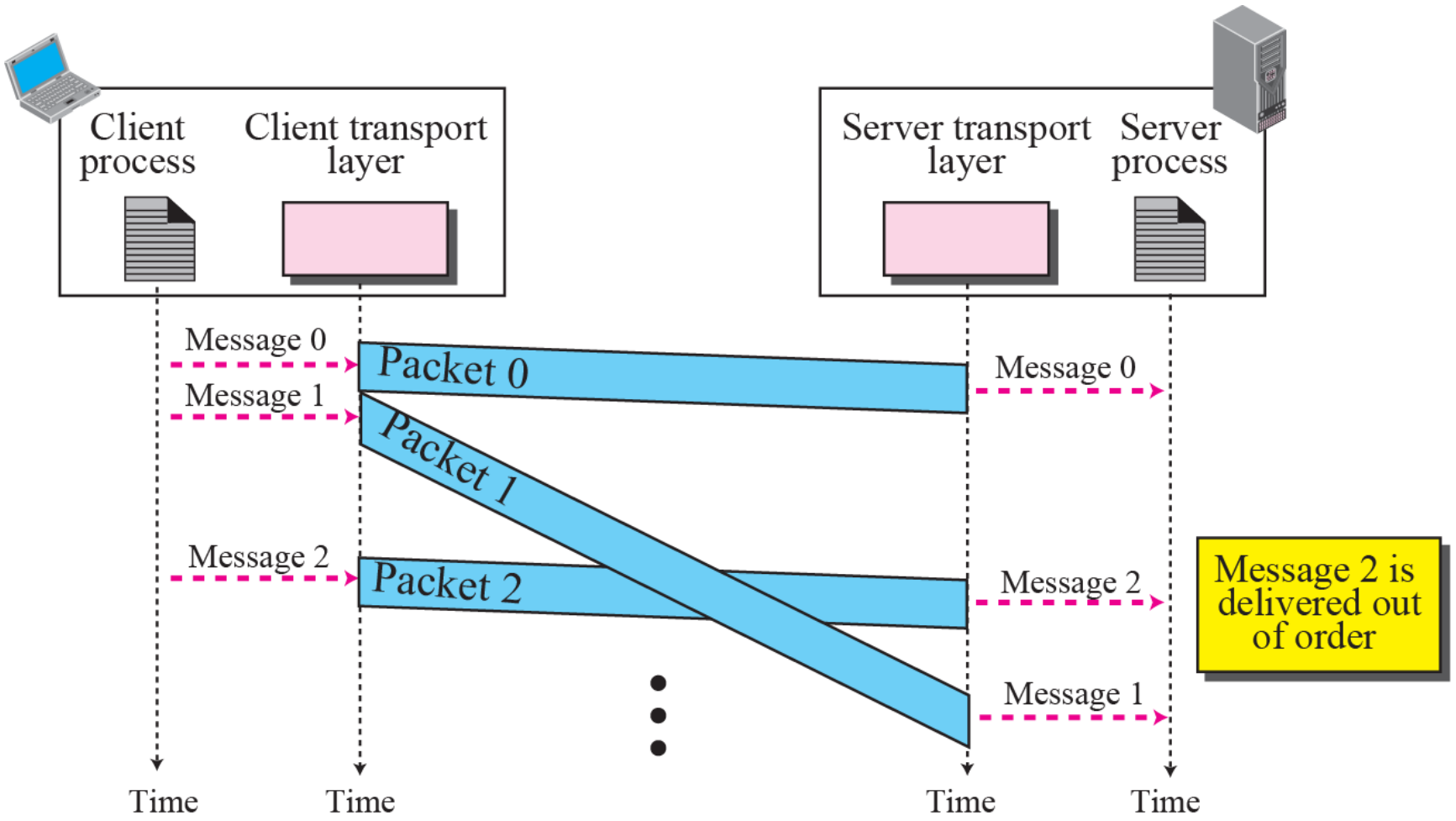


Figure 13.14 *Connection-oriented service*

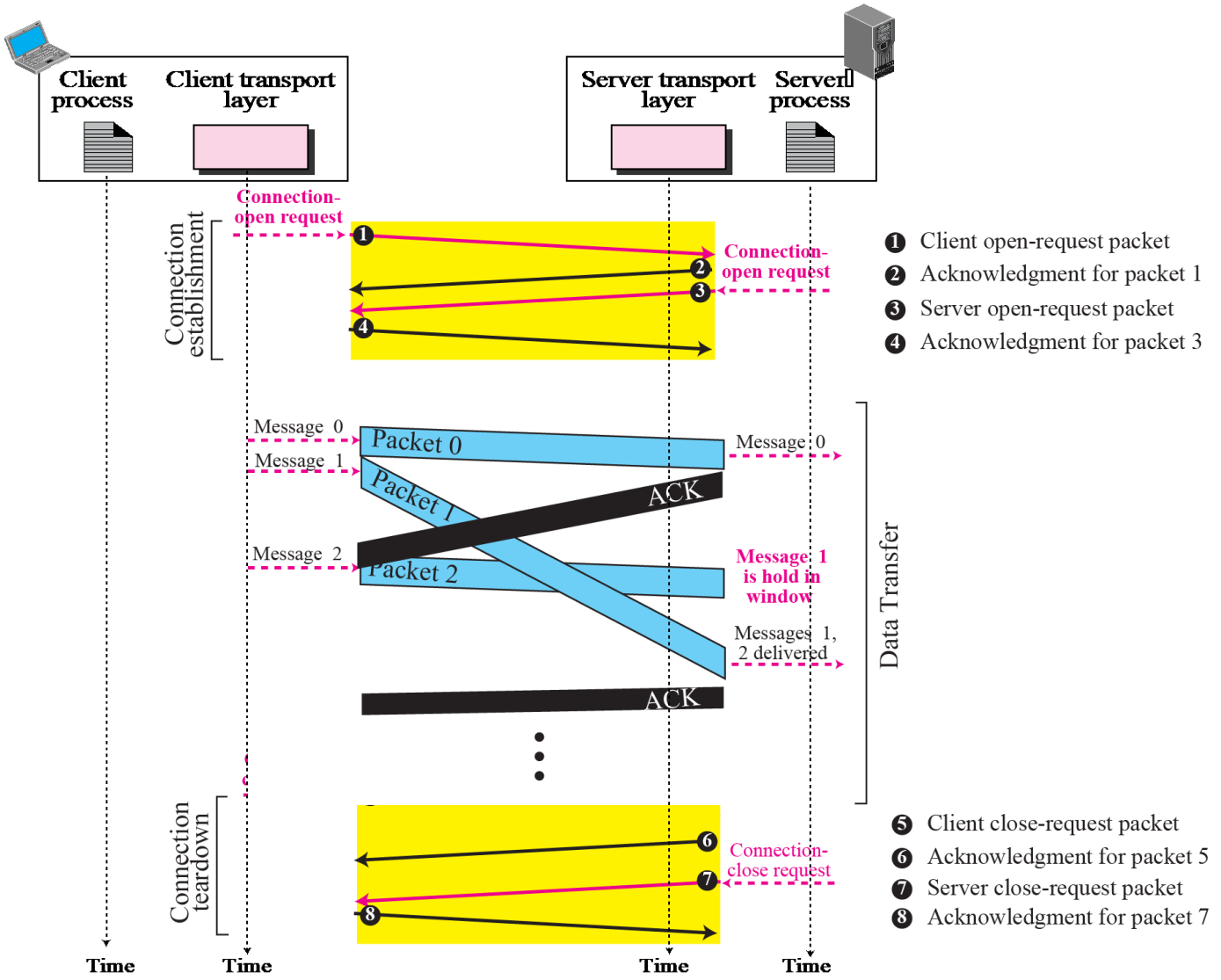


Figure 15.1 *TCP/IP protocol suite*

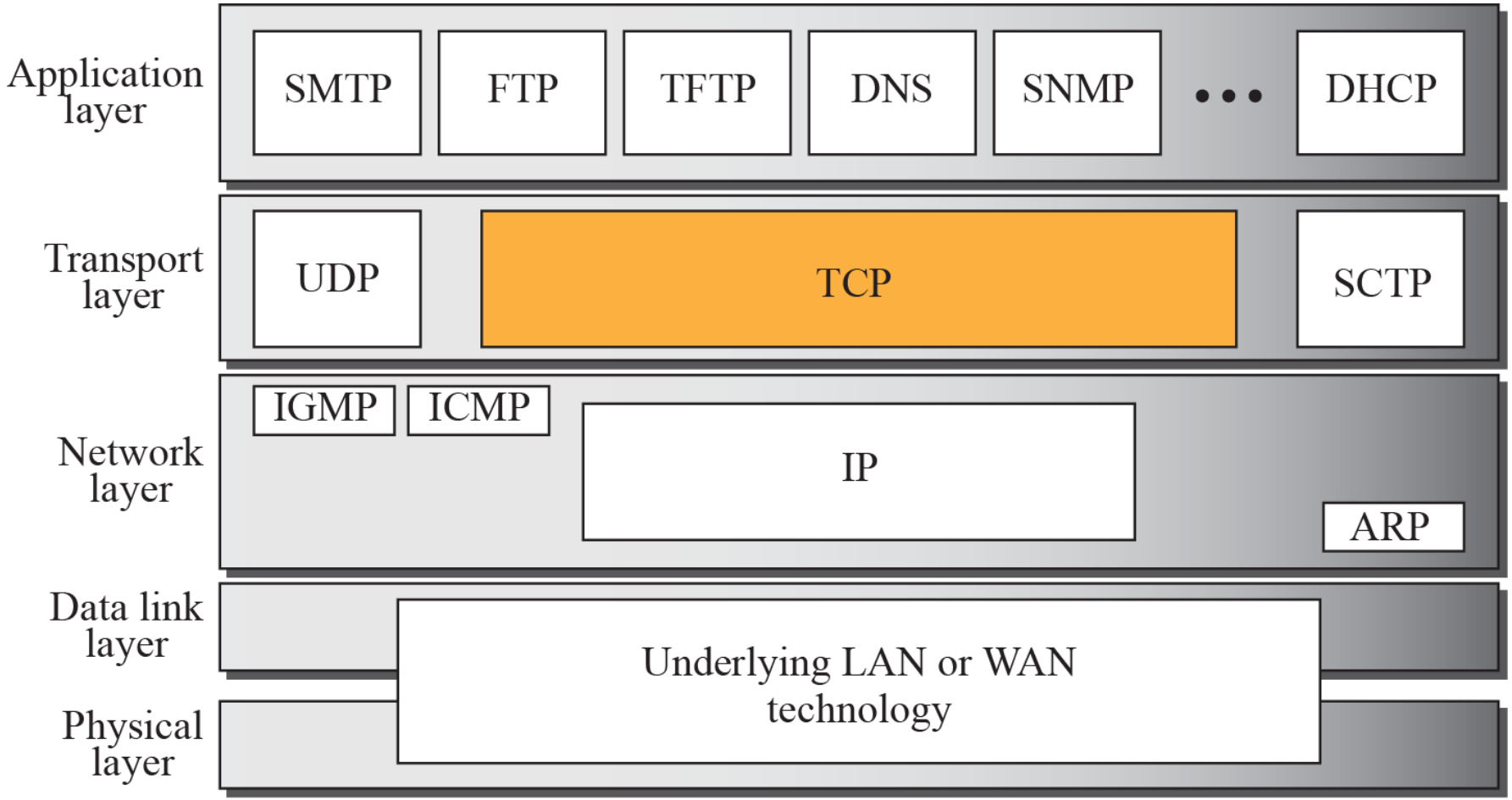


Table 15.1 *Well-known Ports used by TCP*

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
20 and 21	FTP	File Transfer Protocol (Data and Control)
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol

Figure 15.2 *Stream delivery*

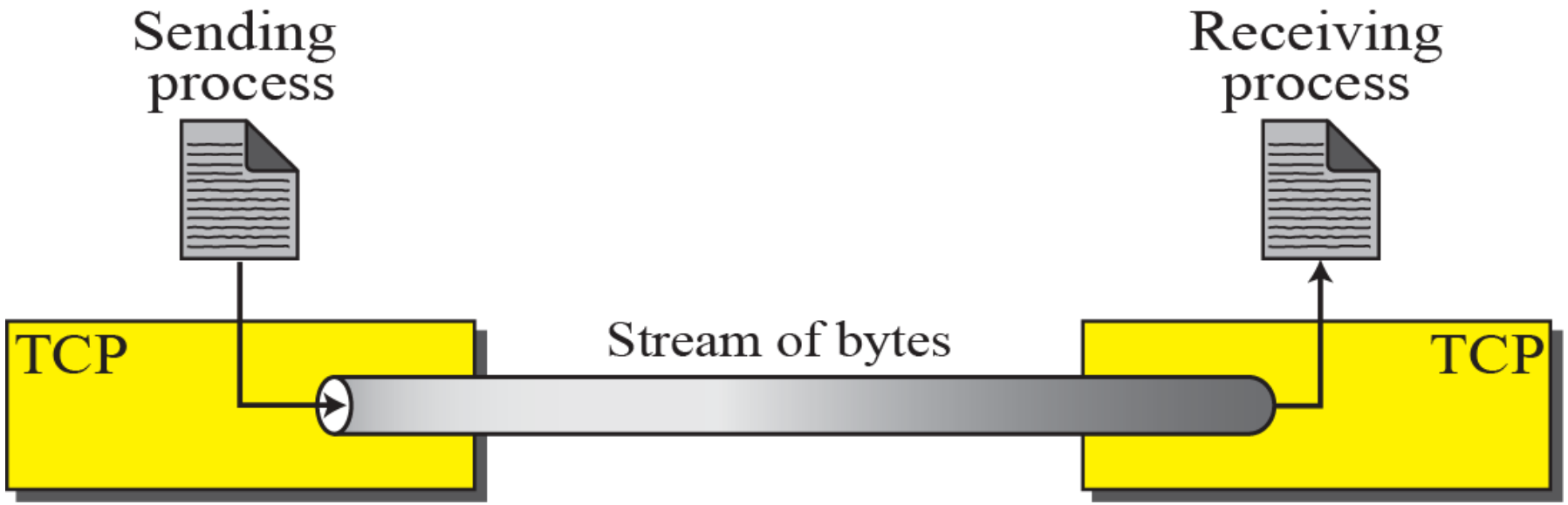


Figure 15.3 *Sending and receiving buffers*

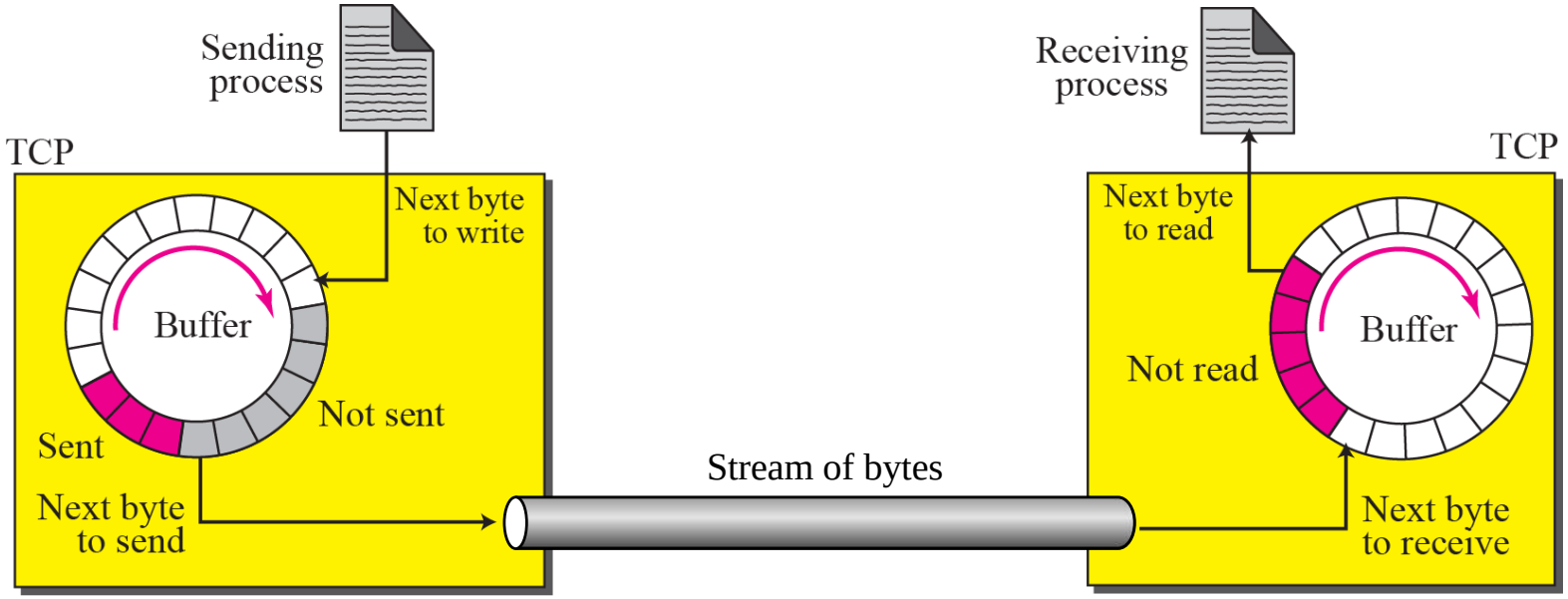
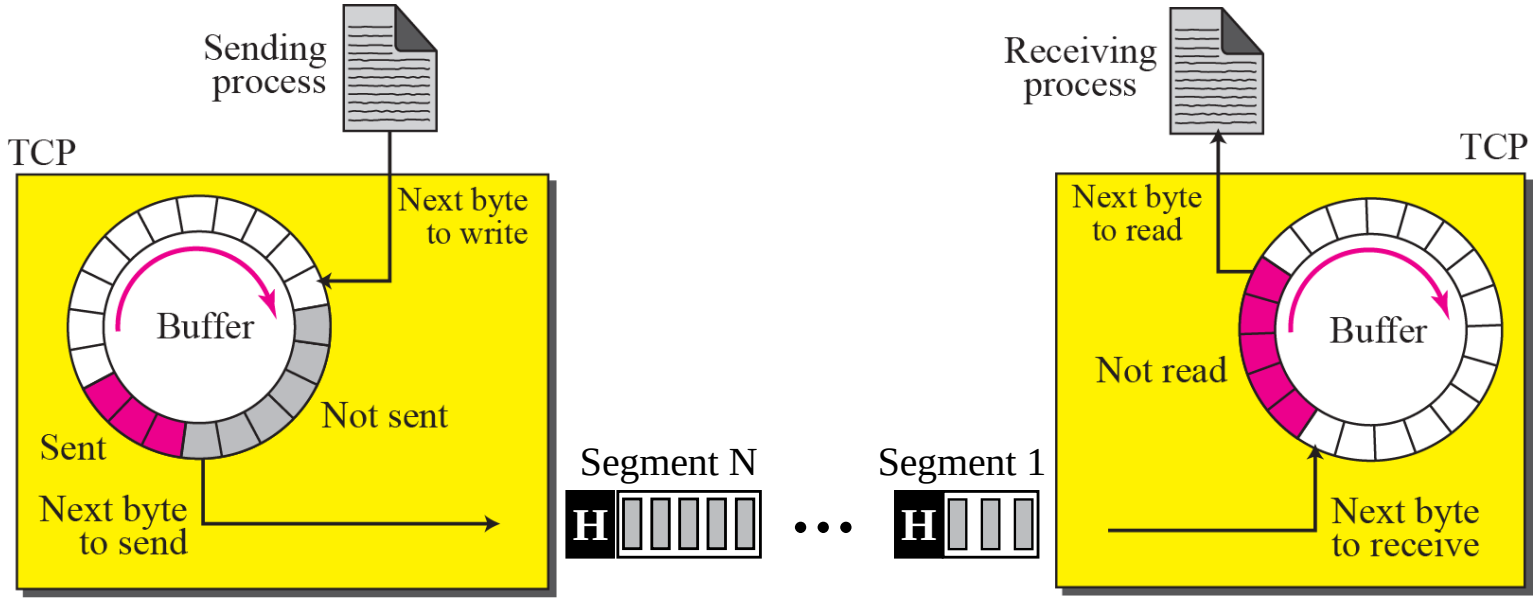


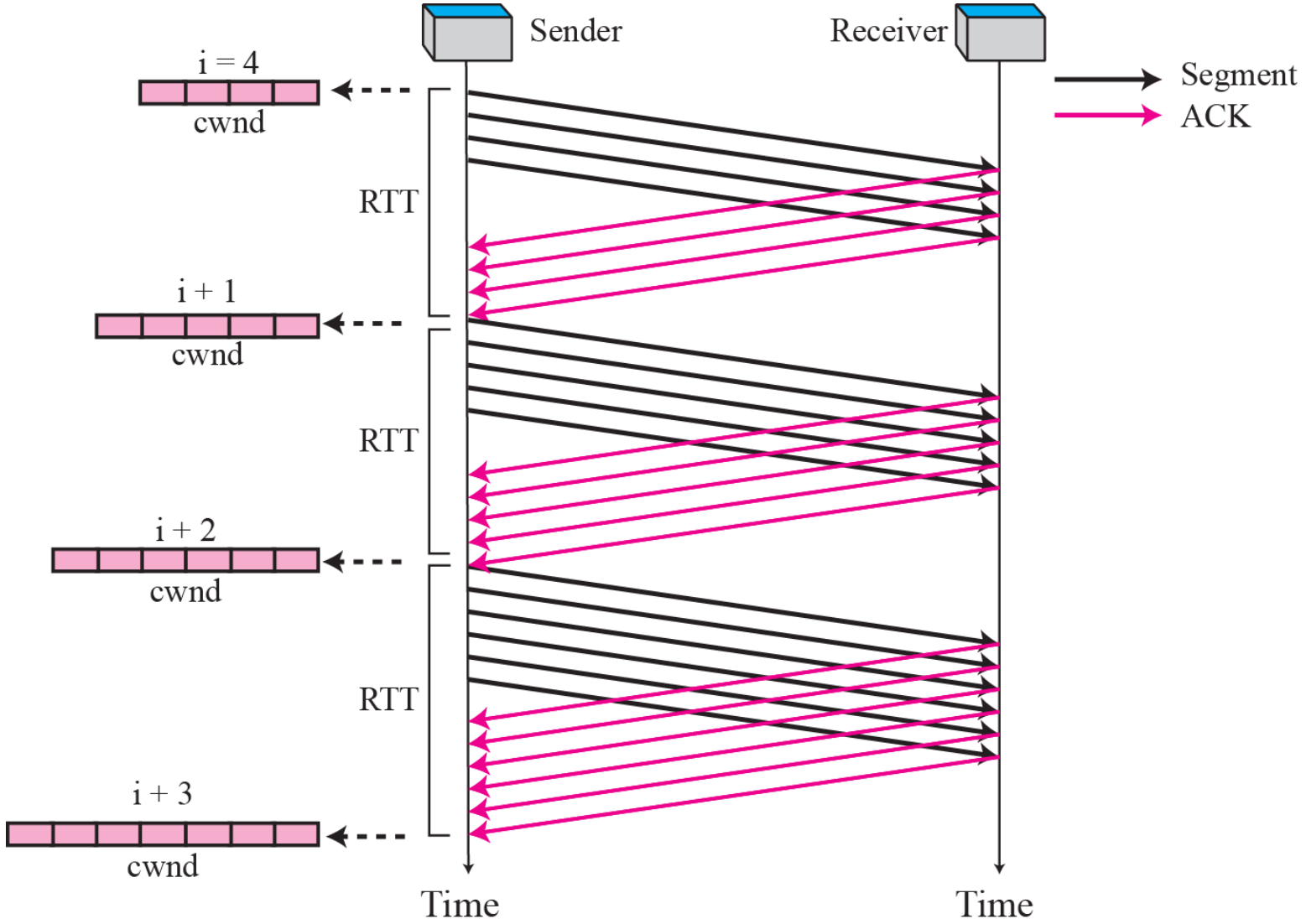
Figure 15.4 *TCP segments*



Note

In the slow start algorithm, the size of the congestion window increases exponentially until it reaches a threshold.

Figure 15.35 *Congestion avoidance, additive increase*



Note

In the congestion avoidance algorithm the size of the congestion window increases additively until congestion is detected.

Figure 15.36 *TCP Congestion policy summary*

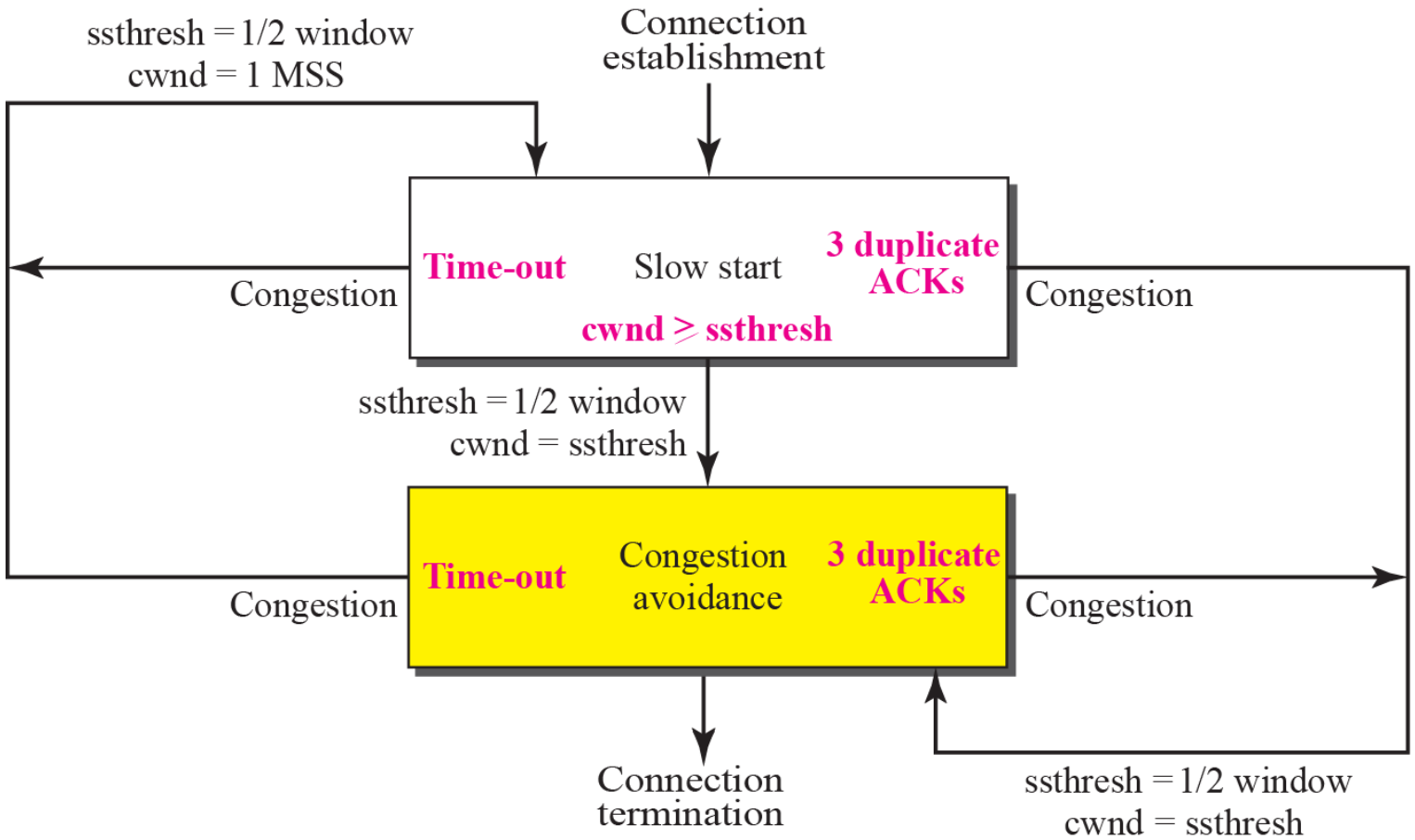
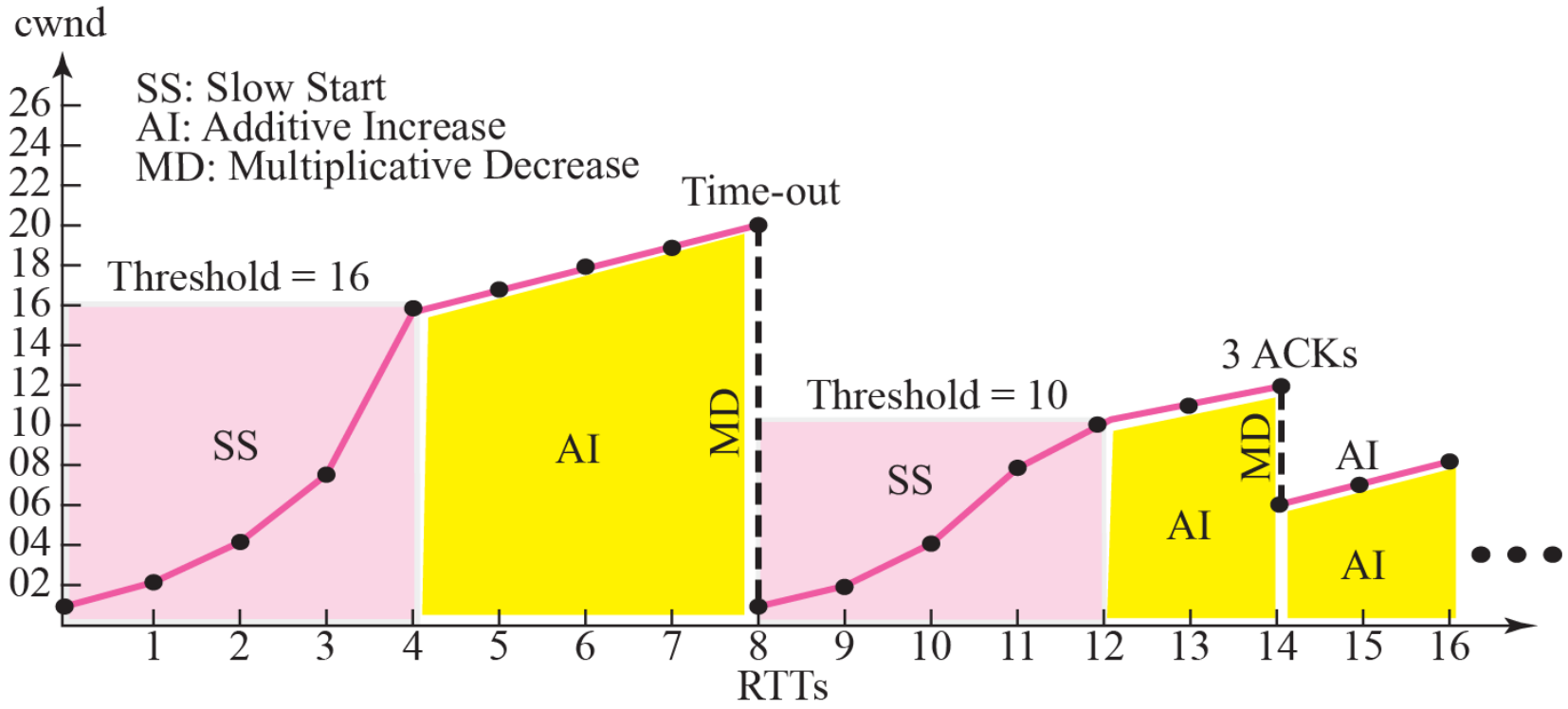


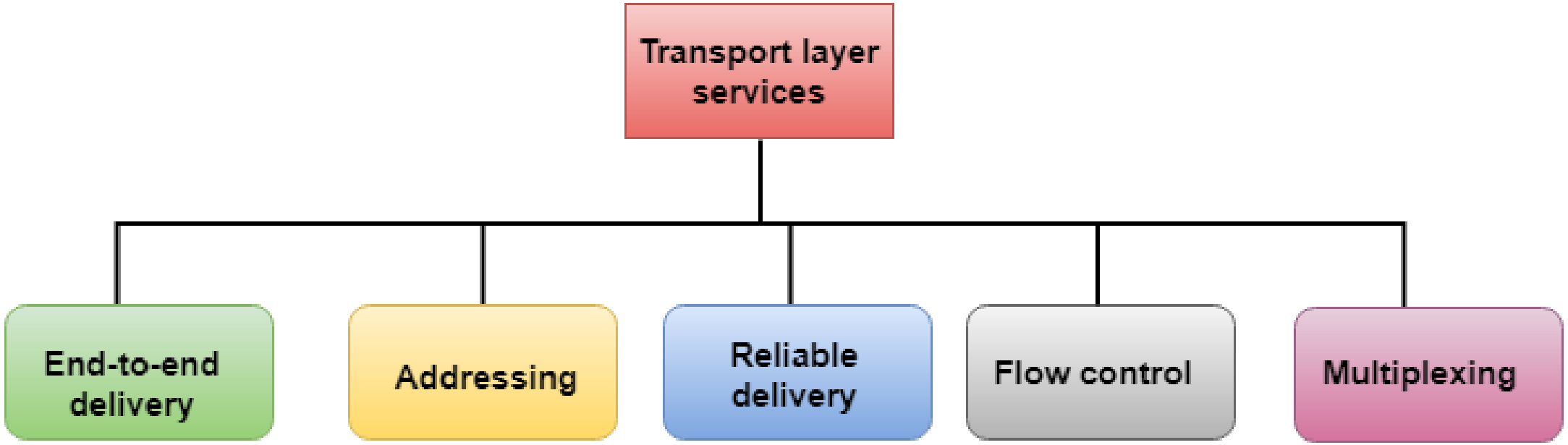
Figure 15.37 *Congestion example*



Services provided by the Transport Layer

- The services provided by the transport layer are similar to those of the data link layer.
- **The services provided by the transport layer protocols can be divided into five categories:**
- **End-to-end delivery**
- **Addressing**
- **Reliable delivery**
- **Flow control**
- **Multiplexing**

Services provided by the Transport Layer

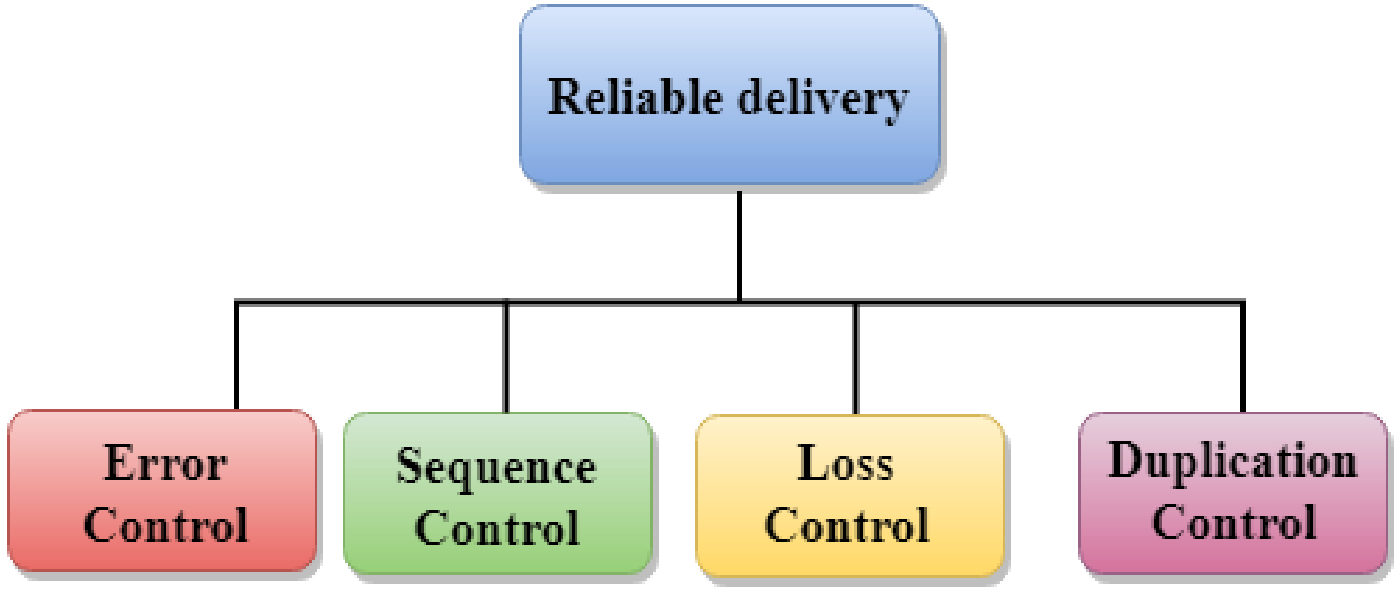


Transport Layer Service – End to End Delivery

- End-to-end delivery: The transport layer transmits the entire message to the destination. Therefore, it ensures the end-to-end delivery of an entire message from a source to the destination.
- Is also called as Port to Port Delivery or Process to Process Delivery

Transport Layer Service - Reliability

- Reliable delivery: The transport layer provides reliability services by retransmitting the lost and damaged packets.
- **The reliable delivery has four aspects:**
- Error control
- Sequence control
- Loss control
- Duplication control



Transport Layer Service - Reliability

- **Error Control :** The primary role of reliability is **Error Control**. In reality, no transmission will be 100 percent error-free delivery. Therefore, transport layer protocols are designed to provide error-free transmission.
- The data link layer also provides the error handling mechanism, but it ensures only node-to-node error-free delivery.
- The transport layer performs the checking for the errors end-to-end to ensure that the packet has arrived correctly.

Transport Layer Service - Reliability

- **Sequence Control :** The second aspect of the reliability is sequence control which is implemented at the transport layer.
- On the sending end, the transport layer is responsible for ensuring that the packets received from the upper layers can be used by the lower layers.
- On the receiving end, it ensures that the various segments of a transmission can be correctly reassembled.

Transport Layer Service - Reliability

- **Loss Control :** Loss Control is a third aspect of reliability.
- The transport layer ensures that all the fragments of a transmission arrive at the destination, not some of them.
- On the sending end, all the fragments of transmission are given sequence numbers by a transport layer. These sequence numbers allow the receivers transport layer to identify the missing segment.

Transport Layer Service - Reliability

- **Duplication Control :** Duplication Control is the fourth aspect of reliability.
- The transport layer guarantees that no duplicate data arrive at the destination. Sequence numbers are used to identify the lost packets; similarly, it allows the receiver to identify and discard duplicate segments.

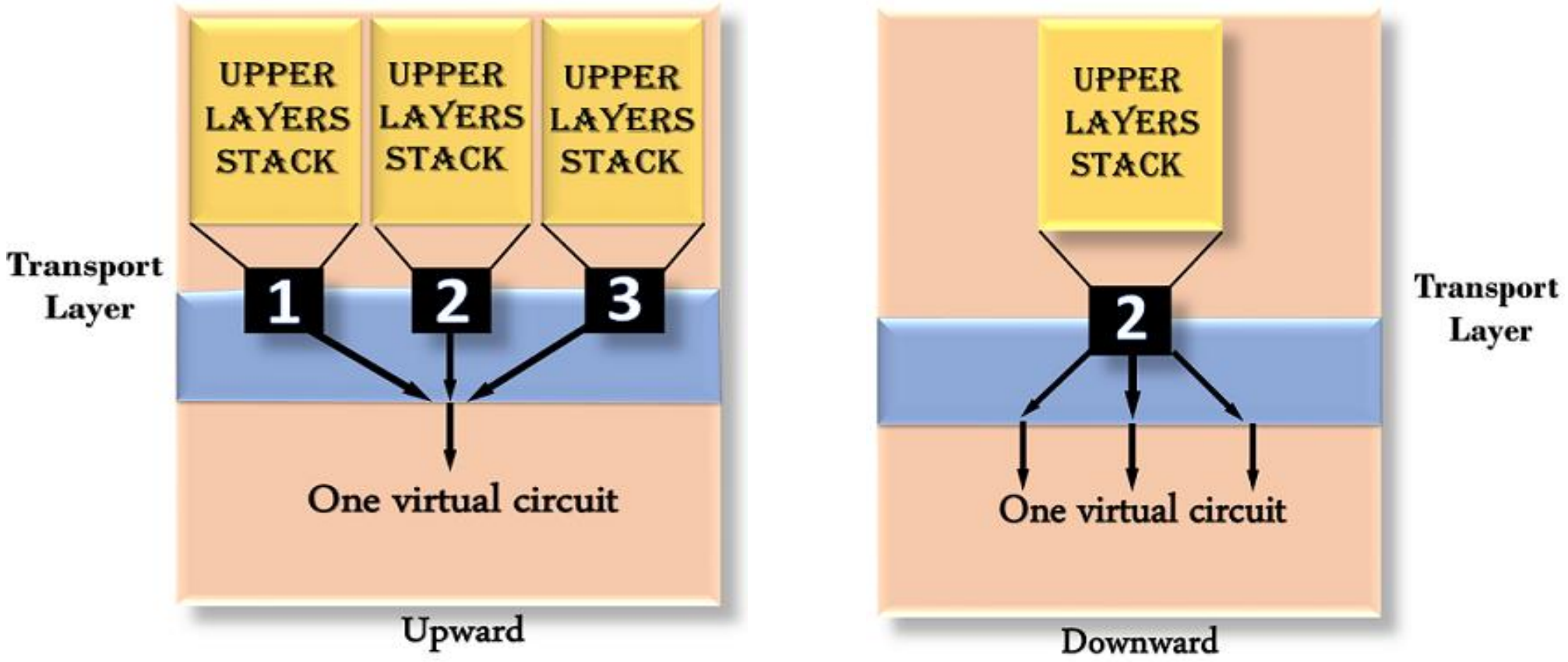
Transport Layer Service - Flow Control

- **Flow control** is used to prevent the sender from overwhelming the receiver.
- If the receiver is overloaded with too much data, then the receiver discards the packets and asking for the retransmission of packets. This increases network congestion and thus, reducing the system performance.
- The transport layer is responsible for flow control. It uses the sliding window protocol that makes the data transmission more efficient as well as it controls the flow of data so that the receiver does not become overwhelmed. Sliding window protocol is byte oriented rather than frame oriented.

Transport Layer Service - Multiplexing

- The transport layer uses the multiplexing to improve transmission efficiency. **Multiplexing can occur in two ways:**
- **Upward multiplexing:** Upward multiplexing means multiple transport layer connections use the same network connection. To make more cost-effective, the transport layer sends several transmissions bound for the same destination along the same path; this is achieved through upward multiplexing
- **Downward multiplexing:** Downward multiplexing means one transport layer connection uses the multiple network connections. Downward multiplexing allows the transport layer to split a connection among several paths to improve the throughput. This type of multiplexing is used when networks have a low or slow capacity

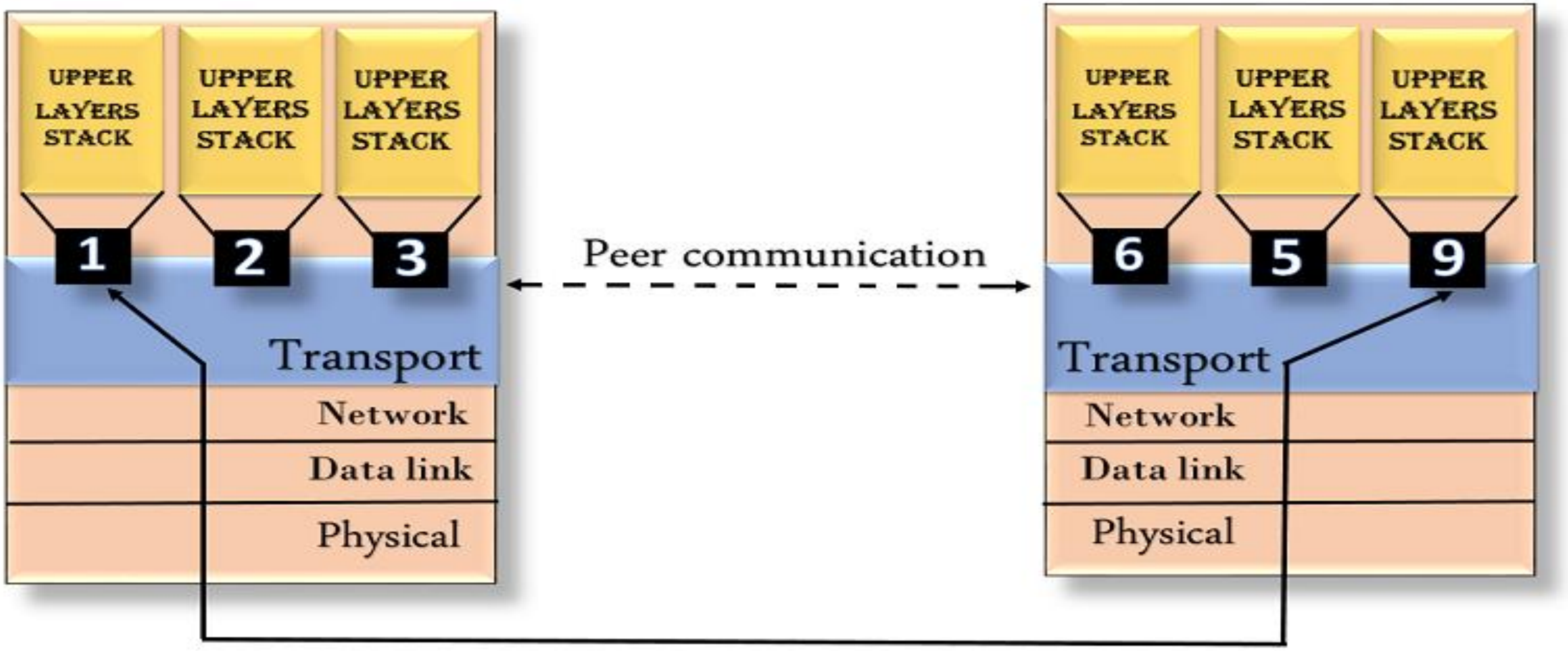
Transport Layer Service - Multiplexing



Transport Layer Service - Addressing

- Data generated by an application on one machine must be transmitted to the correct application on another machine. In this case, addressing is provided by the transport layer.
- The transport layer provides the user address which is specified as a station or port. The port variable represents a particular TS user of a specified station known as a Transport Service access point (TSAP). Each station has only one transport entity.
- The transport layer protocols need to know which upper-layer protocols are communicating

Transport Layer Service - Addressing



Socket

- Definition: A **socket** is one endpoint of a two-way communication link between two programs running on the **network**.
- A **socket** is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.
- An endpoint is a combination of an IP address and a port number.

Socket Programming

- A **socket programming** interface provides the routines required for inter-process communication between applications, either on the local system or spread in a distributed,
- **TCP**/IP based network environment. Once a peer-to-peer **connection** is established, a **socket** descriptor is used to uniquely identify the **connection**.

- **Socket Programming** using TCP/IP [HackerEarth].
- **Socket** programs are **used** to communicate between various processes usually running on different systems.
- It is mostly **used** to create a client-server environment. It provides the various functions **used** to create the server and client **program** and an **example program**.

- Both **Socket and Port** are the terms used in Transport Layer.
- A **port** is a logical construct assigned to network processes so that they can be identified within the system.
- A **socket** is a combination of **port** and IP address.

Berkeley sockets

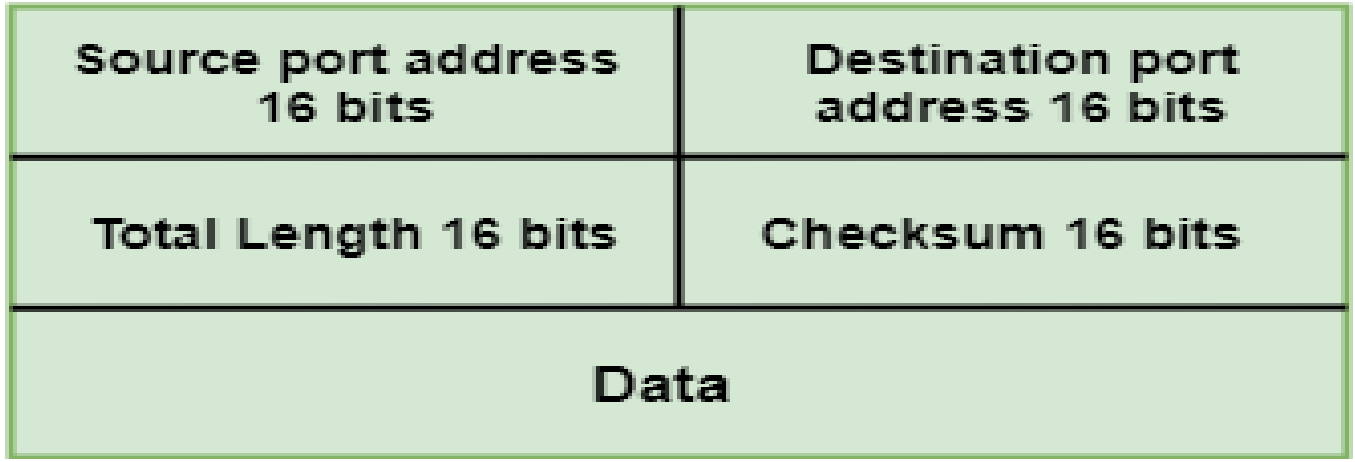
- **Berkeley sockets** is an application programming interface (API) for Internet sockets and Unix domain sockets, used for inter-process communication (IPC). It is commonly implemented as a library of linkable modules. It originated with the 4.2BSD Unix operating system, released in 1983.
- A socket is an abstract representation (handle) for the local endpoint of a network communication path. The Berkeley sockets API represents it as a file descriptor (file handle) in the Unix philosophy that provides a common interface for input and output to streams of data.
- Berkeley sockets evolved with little modification from a *de facto* standard into a component of the POSIX specification. The term **POSIX sockets** is essentially synonymous with *Berkeley sockets*, but they are also known as *BSD sockets*, acknowledging the first implementation in the Berkeley Software Distribution

Transport Layer protocols - UDP

- UDP stands for **User Datagram Protocol**.
- UDP is a simple protocol and it provides non sequenced transport functionality.
- UDP is a connectionless protocol.
- This type of protocol is used when reliability and security are less important than speed and size.
- UDP is an end-to-end transport level protocol that adds transport-level addresses, checksum error control, and length information to the data from the upper layer.
- The packet produced by the UDP protocol is known as a user datagram.

Transport Layer protocols - UDP

UDP Format: The user datagram has a 16-byte header which is shown below:
Source port address: It defines the address of the application process that has delivered a message it is of 16 bits address.
•**Destination port address:** It defines the address of the application process that will receive the message,16-bit address.
•**Total length:** It defines the total length of the user datagram in bytes. It is a 16-bit field.
•**Checksum:** The checksum is a 16-bit field which is used in error detection.



UDP

- Disadvantages of UDP protocol :
- UDP provides basic functions needed for the end-to-end delivery of a transmission.
- It does not provide any sequencing or reordering functions and does not specify the damaged packet when reporting an error.
- UDP can discover that an error has occurred, but it does not specify which packet has been lost as it does not contain an ID or sequencing number of a particular data segment.

TCP - Transmission Control Protocol.

- It provides **full transport layer services to applications**.
- It is a **connection-oriented protocol means** the connection established between both the ends of the transmission.
- For creating the connection, TCP generates a virtual circuit between sender and receiver for the duration of a transmission.

Features Of TCP protocol

- **Stream data transfer:** TCP protocol transfers the data in the form of contiguous stream of bytes. TCP group the bytes in the form of TCP segments and then passed it to the IP layer for transmission to the destination. TCP itself segments the data and forward to the IP.
- **Reliability:** TCP assigns a sequence number to each byte transmitted and expects a positive acknowledgement from the receiving TCP. If ACK is not received within a timeout interval, then the data is retransmitted to the destination.
  The receiving TCP uses the sequence number to reassemble the segments if they arrive out of order or to eliminate the duplicate segments.

Features Of TCP protocol

- **Flow Control:** When receiving TCP sends an acknowledgement back to the sender indicating the number the bytes it can receive without overflowing its internal buffer. The number of bytes is sent in ACK in the form of the highest sequence number that it can receive without any problem. This mechanism is also referred to as a window mechanism.
- **Multiplexing:** Multiplexing is a process of accepting the data from different applications and forwarding to the different applications on different computers. At the receiving end, the data is forwarded to the correct application. This process is known as demultiplexing. TCP transmits the packet to the correct application by using the logical channels known as ports.

Features Of TCP protocol

- **Logical Connections:** The combination of sockets, sequence numbers, and window sizes, is called a logical connection. Each connection is identified by the pair of sockets used by sending and receiving processes.
- **Full Duplex:** TCP provides Full Duplex service, i.e., the data flow in both the directions at the same time. To achieve Full Duplex service, each TCP should have sending and receiving buffers so that the segments can flow in both the directions. TCP is a connection-oriented protocol. Suppose the process A wants to send and receive the data from process B. The following steps occur:
  - Establish a connection between two TCPs.
  - Data is exchanged in both the directions.
  - The Connection is terminated.

References

- Computer Network Tutorials Videos Gate Smashers
- Javatpoint